

MSRBDMS.NET User's Help Manual



Original Release: July 2009

Updated: October 2009

Updated: March 2010

Updated: September 2012

Updated: May 2013

Updated: September 2013

Contents

Getting Started	1
Welcome to RBDMS.NET	1
Hardware and Software Requirements	1
Installation and Setup	2
Database Connections	2
Application Configuration and Settings	3
Customizing RBDMS.NET	8
Managing Codes and Reference Tables	8
To add a new master code group, ...	8
To add codes to a master code group, ...	8
Other Reference Tables	9
Mississippi-specific RBDMS.NET Customizations	9
5% POFR Well Fees Manager	9
Unplugged Well Fees Manager	11
Void Receipt Procedure	12
Perforations vs. Formations	12
Inspector Search and Assignment	13
PermitApprove Right	13
7520 Reports	13
System vs. user Dates	14
Adding Reports	15
Available Report Formats	15
Defining Reports	17
Creating a Report Project	18
Adding a Report to the Project	19
Specifying the Dataset for Binding	19
Adding Header and Footer Information	19
Developing the Report Body	20
Fixing Undesirable Page Breaks	23
Creating Report Parameters	23
Creating and Associating Filters	24
Adding Reports to the Application Menu	27
Testing and Debugging Reports, Filters, and Menus	27
A Word about Subreports	27
Integration with Other Applications	28
Links to eReport.NET	28
Links to LaserFiche	28
Links to eForm	29
Links to Log Images	29
Microsoft Office CiceroUIWndFrame	30
Managing RBDMS.NET Security	31
Using RBDMS WinAdmin.NET	31
Managing the Production Environment	31
Backing Up, Restoring, and Maintaining the RBDMS.NET Database	31
Replicating the Database	33
Configuring Source Code	33
Working with the Dynamic User Interface	36
About the Dynamic User Interface	36
DynUI File Folder	36
DynUI XML Files	36
General Description of the RBDMS.NET User Interface	36
About the DynUI XML Structure	39
RBDMS_UI_Description	39
SubTabOrder: Space-delimited String	39

OneToOneRelationsToDisplay: Comma-delimited String.....	39
RemoveObservableCollections: Comma-delimited String.....	40
Form View	40
FormGrid Definition.....	42
Form Field Descriptions	42
List View	45
ListColumn Definition	46
List Field Descriptions.....	47
Dynamic UI Element Types	49
About Dynamic UI Element Types	49
Attributes Common to All DynUI Elements	49
Label.....	50
TextBox.....	51
ReadOnly TextBox	51
Multiline TextBox	52
ComboBox	53
TimePicker.....	54
Calendar ComboBox.....	55
TextBlock	56
DeleteButton.....	56
FormExpanderButton	57
SaveButton.....	57
Print Button	58
Button.....	58
NavigateButton	59
CheckBox	59
Pick-and-View	60
SimpleStackPanel	62
ReportGrid	63
Generic XAML Elements	64
VisualBasic.NET Programmer's Reference	65
When Recompiling Is Needed	65
Adding, Editing, or Deleting Business Logic.....	71
Change Tracking	72

Getting Started

Welcome to RBDMS.NET

The Mississippi State Oil and Gas Board (MSOGB) and the [Ground Water Protection Council](#) (GWPC) welcome you to RBDMS.NET version 2.0. This administrator's help file contains information useful to database administrators and developers for configuring the application and extending its functionality through report additions, code management, judicious user interface editing, and data management. A more general user's manual geared toward data entry and management staff also has been prepared for the MSRBDM.NET software application. Technical and administrative staff should be familiar with the content of both manuals, since each manual is intended to complement the other.

This manual is meant to document the programming done to support the RBDMS.NET application. It is not meant to be a training manual in SQL Server administration procedures or working in Visual Studio.NET, nor is it a substitute for the extensive [Microsoft](#) help system.

With this release of the MSOGB's installation of RBDMS.NET, the GWPC has completed a major update of its Risk Based Data Management System (RBDMS) to .NET. This new application includes a powerful search function, GIS, and highly granular security in a .NET application suitable for use within agency offices and for field inspection. The front-end application is paired with a SQL Server 2008 R2 database that houses data associated with well and well history tracking (ownership, bonding, permitting, location, construction, inspection, production, and plugging/restoration), compliance monitoring, and related financial transactions.

Systems analysis and .NET programming for this project were provided by [Coordinate Solutions, Inc.](#) and [Virtual Engineering Solutions, Inc.](#)

Hardware and Software Requirements

Hardware specifications for the server should support the following prerequisites for running MSRBDM.NET:

1. Microsoft .NET Framework 4.0: <http://www.microsoft.com/en-us/download/details.aspx?id=17851>.
2. SQL Server 2008 R2 or 2012 (with advanced services)

While not required to run MSRBDM.NET alone, servers running the RBDMS Data Mining application also should support Internet Information Services 7.0 or higher. Internet Explorer version 9 or other up-version browser is recommended for client machines.

Other software that will be useful for database administrators include either of the following applications for designing .rdlc templates as extensions for the RBDMS Reports module:

- Visual Studio.NET 2010 or 2012

Release software includes the following deliverables:

1. MSRBDM.NET.bak:

<http://www.rbdmsonline.org/downloads/MS/MSRBDMS.zip>

2. MSRBDMS.NET client .msi installer:
http://RbdmsOnline.org/downloads/MS/RbdmsWpf_MS.zip
3. RBDMSSecurity.bak:
<http://www.rbdmsonline.org/Downloads/MS/RBDMSSecurity.zip>
4. RBDMS WinAdmin.NET, used to manage application security, menus, report filters. and rule sets:
<http://RbdmsOnline.org/downloads/MS/RbdmsWinSetup.zip>
5. MS GIS Vector Data: This data is maintained by MSOGB staff.

Client workstations should likewise be adequate for supporting the .NET Framework 4.0. RBDMS.NET was developed to run on XGA (1024 x 768) monitors or better.

Installation and Setup

Database Connections

Installing MSRBDMS.NET on the server is a matter of the following steps:

1. [Restore the MSRBDMS.bak and RbdmsSecurity.bak](#) to the server on the local area network.
2. Install the front-end application on the client workstations by running the .msi installer and stepping through the wizard.
3. Connect the client to the SQL Server via modified database connections, as follows:
 - a. Launch RBDMS.NET by clicking **Start | All Programs | GWPC | Rbdms.Net MS**.
 - b. Open the **Database Connections** form by clicking the **Configure** button at the bottom of the login page. Note that this form also can be opened after login from the main menu (**Configuration Activities | Database connection settings**).
 - c. Modify, test, and save each connection string (RBDMS Data, Security Database, and DMZ Database) by clicking the appropriate radio button, modifying the connection information, clicking **Test Connection**, and then **Save Connection**. Settings take effect when RBDMS.NET is re-opened.
 - d. Login using either Windows or ASP.NET security.
 - a. Windows – click **Login** under Windows.
 - b. ASP.NET – enter your user name and password and click **Login** under ASP.NET.

For more information about managing RBDMS users within SQL Server, please see [Managing the Production Environment](#).

Application Configuration and Settings

Database settings and user settings are separated in RBDMS.NET, which allows users to configure certain preferences for their views of the information within their own login, while providing a consistent user interface for all users.

Settings that are specific to an assigned user name are controlled by the **user.config** file buried deep in the domain user's **/AppData** folder (For example, **C:\Users\dgillesp.VES\AppData\Local\CS_and_VES\RbdmsWpf.exe_Url_vlb325ax0yje1u4t15iyh3sihdcs15lj\0.3.1.1.**). This file typically controls such settings as the *ReportPath*, *GISHomePage*, and *GISShapePath*. These paths typically can be set on the **Configuration Activities | User configuration settings** page of RBDMS.NET.

However, for MSRBDM.NET, at the Agency's request, all user settings have been changed to application settings except for *UserConnectionString*, *UserSecurityConnectionString*, and *UserDMZConnectionString*. These must be user settings to facilitate certain encryption operations in other RBDMS installations nationally. However, this should not affect MSRBDM.NET since they are set to the application settings at run time. However, the RBDMS developers strongly suggest that the MSOGB database administrator remove the **User configuration settings** menu item from general users' installations through RBDMS WinAdmin.NET to avoid the security issue of having clear-text database connection strings showing in these user settings.

The **RbdmsWpf.exe.config** file controls numerous application-wide settings. At runtime, this file is stored in the installation folder, which is by default **C:\Program Files\GWPC\Rbdms.Net**.

The main program settings found here are as follows. Note that not all of these settings are used in every agency installation of RBDMS.NET.

AllowDomainLogin(Application)

If true then the application allows users to be authenticated using their current domain login information. This option is usually only true for an intranet application. Set to *True* in MSRBDM.NET.

AllowFormLogin(Application)

If true then the application will allow users to be authenticated using forms authentication. Set to *True* in MSRBDM.NET.

Appdirectory(User)

The application directory.

ApplicationName(Application)

The name of the application. Used to retrieve application specific settings from the form based security database. For Mississippi, this is set to "MSRBDM.NET"

AutoSave(User)

Saves all user settings if any one of them are changed in the application.

Set to *True* in Mississippi.

ChangeTrackerTrackPreviousValue(Application)

Gets or sets a value indicating whether the change tracker saves the TO or the FROM value of a change. For use by developers and administrators. *True* means the FROM value, *False(default)* means the TO value.

CommandTimeout(User)

This controls the amount of time that the program waits for results from the database when executing SQL for a report, in seconds. Maximum timeout in seconds for SQL commands in the application. Set to *120* in MSRBDM.NET.

DefaultPage(User)

Set to *~\Default.aspx* in MSRBDM.NET.

GisDetachedConfig(User)

The GIS detached config file name. Unused in MSRBDM.NET.

GisHomePage(Application)

Whether the GIS is displayed in the **Home** page of the application. Set to *True* in MSRBDM.NET.

GisOutputPath(User)

The temporary directory where the GIS puts rendered maps before they are displayed on the screen. This path must have read/write permissions. If the path exists as is, then it is considered to be an absolute path. Otherwise it is considered a relative path to the location of the **RBDMSWPF.exe** executable. If the path does not exist, an error is thrown. Set to the relative path *\ms\output* in MSRBDM.NET.

GisShapePath(User)

The directory where the GIS looks for the .SHP GIS files. This path must have read permissions. If the path exists as is, then it is considered to be an absolute path. Otherwise, it is considered a relative path to the location of the **RBDMSWPF.exe** executable. If the path does not exist, an error is thrown. Set the relative path *"\data\ms"* in the desktop version of MSRBDM.NET.

GisUrl(User)

The GIS URL. Unused in MSRBDM.NET.

GisViewer(User)

The GIS viewer. In MSRBDM.NET, set to "InternetViewer."

Inspector(User)

The default inspector name for this computer. In MSRBDM.NET, set to *Unk*.

KeyType(User)

The key type (string or integer) used for the location ID in the GIS. Set to *Integer* in MSRBDM.NET

LastWhere(User)

The last SQL WHERE clause.

LocLayer(User)

The name of the GIS layer containing the location ID used for navigation. Set to *RBDMS Wells* in MSRBDM.NET.

MaxHoleNumber(User)

The maximum number of holes in a given county. In MSRBDM.NET, set to 79999.

NewIDTimeout(User)

Gets or sets the timeout in hours for New IDs (i.e., API numbers, permit numbers, etc.). Primarily used when generating a new unique API number. This API will be entered into the *RefNewID* and reserved for 24 hours, or whatever the .config file setting is for *NewIDTimeout*. Set to 24 hours in MSRBDM.NET.

ProjectType(User)

The type of the project (i.e., RBDMS, Mine, etc.). Set to *RBDMS*.

RBDMSConnectionString (Application)**RbdmsSecurityConnectionString****RbdmsDmzConnectionString****RBDMSConnectionStringTEST****RbdmsSecurityConnectionStringTEST****RbdmsDmzConnectionStringTEST**

Connection strings to the database with the RBDMS tables.

RbdmsWcfService(User)

The RBDMS WCF service path. Unused in MSRBDM.NET.

RecordCount(User)

Gets or sets the maximum record count to retrieve for reports. Set to 100 in MSRBDM.NET.

ReportPath(User)

This sets the location of where the program looks for the report RDLC and report XML files. If it is an empty string, then the program will look for these files in the "Reports" folder underneath the location of the RBDMS.NET executable (**RbdmsWpf.exe**).

StateName(User)

The name of the State. Set to *Mississippi* in MSRBDM.NET.

StateNumber(User)

The FIPS number of the State. Set to 23 in MSRBDM.NET. Used when generating a new API number.

TestMode(Application)

Sets whether the application should run in TEST Mode (*True*) or Production mode (*False*). Test mode changes the colors of the buttons on the left side of the application to orange instead of blue in order for a user to distinguish which version he or she is using. Also, the TEST connection strings are used in encryptConnectionStrings3DES. Finally, a debug dialog is shown at the beginning of the application, indicating the location of the reports and GisShapePath. Not present in MSRBDM.NET.

UserConnectionString(User)**UserSecurityConnectionString****UserDMZConnectionString**

Override the specified connection strings.

WebDirectory(User)

Directory path to the ASP.Net web directory. Unused in MSRBDM.NET.

Imaging service endpoint:

This sets the endpoint for the Imaging service. It is not strictly part of the Application settings per se, but is in the **app.config** file near the end. Look for this in the **app.config** and change the <endpoint> as needed:

```
<client>

<endpoint address="https://servername/ImagingInterface/service.svc"
    binding="wsHttpBinding" bindingConfiguration="Service_svr"

contract="OCC.ImagingInterface.IService2" name="OCCImageService_test" />
```

```
<endpoint address="https://servername/ImagingInterface/service.svc"  
    binding="wsHttpBinding" bindingConfiguration="Service_svr"  
contract="OCC.ImagingInterface.IService" name="OCCImageService" />  
</client>
```

Customizing RBDMS.NET

Managing Codes and Reference Tables

The ability to change codes should be reserved for the system administrator. Adding, updating, or deleting codes should be done with great care. It is possible to make the output of some reports such as the EPA 7520 invalid by improper adjustments to the codes. Additionally, any code-driven combo box in the system will display blank if the data in the database has a value that is not in the codes.

RBDMS.NET makes extensive use of codes to specify types of actions, results of actions, and types of equipment. A master codes table is used for storing many code types and a description of each. Administrators can easily add new types of equipment or testing methods through the RBDMS **Reference Codes** editor, which is available on the **Reference Activities** menu of the RBDMS.NET front-end application.

To add a new master code group,

1. Click the **New** button next to the **Field** combo box.
2. Enter the **Field** name and click **OK**.
3. Complete the remainder of the master code form to note the description, the source, the code type, the tables the code governs, and the maximum length.
4. Create the codes for the master code group.

To delete a master code, click the **Delete** button next to the **Field** combo box. This action cannot be undone unless you restore the RefCode and RefCodes tables from a backup. Therefore, deleting a master code group should be done only after careful consideration of any application functionality or reports that depend on the code group.

To add codes to a master code group,

1. Choose **Reference Activities | Ref Code Editor** from the RBDMS.NET menu.
2. Then choose the appropriate code group from the **Field** combo box.
3. Click **Add** and enter a new code on the new record created. Be sure that the code is the appropriate data type for the associated field. If it is a text field, make sure that it does not exceed the maximum field length.
4. Click the hot key **Alt+a** to advance data entry to the next new record to continue adding successive codes to the code group.

Be sure that the code is the appropriate data type for the associated field. If it is a text field, make sure that it does not exceed the maximum field length. The tag and value are optional fields used to store text and/or numeric information when needed for special codes.

To delete a code from the code group, click the **Delete Row**  icon.

If a code will not be used anymore, turn its *Active* status off. When a code's active flag is off, it will not be available in combo boxes on data entry forms, but it will still appear on reports.

Other Reference Tables

Forms for managing the reference tables for formations, pools, fields, and counties are also available from the **Reference Activities** menu. These grid-style forms can be sorted in ascending and descending order by clicking on the column headers. A **New Record** row is available at the top of the form to accept new entries. You can edit existing entries within the grid.

Mississippi-specific RBDMS.NET Customizations

Some of the business logic specific to the MSOGB's installation of RBDMS.NET is discussed in this section.

- [5% POFR](#) Well Fees Manager
- [Unplugged](#) Well Fees Manager
- [Void Receipt](#) Procedure
- Perforations [vs.](#) Formations
- [Inspector search](#)
- [PermitApprove](#) right
- [7520](#) reports

5% POFR Well Fees Manager

Because the Proof of Financial Responsibility (POFR) rule is based on a Board decision, and the percentage levied or the scale itself might change in the future, the business logic for the 5% POFR Well Fees Manager is set in a stored procedure called *GenerateAnnualWellFee*. It can be changed without modification to the application. The criteria for the well list pulled are as follows:

Not Class 2

AND

Not covered by a bond

AND

Status in (AA,CI,CPL, DG, I, NR, PR, WS)

For each well, the total depth is compared against the following scale per Rule 4 and 5% of that amount is billed to the operator:

- Zero to 10,000: \$20,000
- 10,001 to 16,000: \$30,000
- 16,001 or more: \$60,000

The due date for the fees is derived from the month and day of the first approved

permit on the current year. The following view and stored procedure are included for reference:

```

select distinct
p.wellkey
--, P.FirstApprovedDate
, case when isdate(

cast(month(p.firstapproveddate) as varchar(2)) + '/' +

cast(day(p.firstapproveddate) as varchar(2)) + '/' + cast(year(@inDate) as varchar(4)))=1 then

convert(datetime,

cast(month(p.firstapproveddate) as varchar(2)) + '/' +

cast(day(p.firstapproveddate) as varchar(2)) + '/' + cast(year(@inDate) as varchar(4)))

else

convert(datetime,

cast(month(p.firstapproveddate) as varchar(2)) + '/' +

cast(day(p.firstapproveddate)-1 as varchar(2)) + '/' + cast(year(@inDate) as varchar(4)))

end as DueDate
, case when c.depthbottom is null or c.depthbottom <=10000 then 20000 * 0.05

when c.depthbottom >=10001 and c.depthbottom <=16000 then 30000 * 0.05

when c.depthbottom >=16001 then 60000 * 0.05

end as amount

from

(

select w.PKey as WellKey, max(l.DepthBottom) as DepthBottom
From Well w left join Construct c on w.PKey=c.WellKey

left join Loc l on c.PKey=l.ConstructKey

group by w.PKey

) as c

inner join vwFirstApprovedDate as p on p.WellKey=c.WellKey

inner join Well w on p.WellKey=w.PKey

where (

(month(p.firstapproveddate) < month(@inDate) and YEAR(p.FirstApprovedDate) <= YEAR(@inDate))

```

```

or (month(p.firstapproveddate) = month(@inDate) and day(p.firstapproveddate)
<= day(@inDate) and YEAR(p.FirstApprovedDate) <= YEAR(@inDate))
)

and w.welltype NOT IN ('EOR','SWD')
and w.pkey NOT IN (select distinct wellkey from BondLink)
and w.wellstatus IN ('AA','CI','CPL','DG','I','NR','PR','WS')
and w.pkey NOT IN (
    select wellkey
    from wellannualfee fee
    where fee.WellKey=p.WellKey
and month(fee.duedate) = month(p.firstapproveddate) and YEAR(fee.duedate)=Y
EAR(@inDate)
)

```

vwFirstApprovedDate

```

SELECT
dbo.PermitLink.WellKey, MIN(dbo.PermitDate.EventDate) AS FirstApprovedDate
FROM      dbo.PermitDate INNER JOIN
          dbo.PermitLink ON dbo.PermitDate.PermitKey
          = dbo.PermitLink.PermitKey
WHERE
(dbo.PermitDate.Event = 'PermitStatus') AND (dbo.PermitDate.Comment
= 'Approved')
GROUP BY dbo.PermitLink.WellKey

```

Unplugged Well Fees Manager

The criteria for the well list pulled by the Unplugged Well Fees Manager are as follows:

Wells that are

```

Not Class 2
AND
Not field name = 'WILDCAT'
AND
Status in (AA,CI,CPL, DG, I, NR, PR, WS)

```

The business logic for the Unplugged Well Fees Manager is set in a stored procedure called *GenerateUnpluggedWellFee*. It can be changed without modification to the application. The due date for these fees is derived as June 15 plus the incoming year:

```

declare @thisYear int
set @thisYear=year(@indate)
declare @thisDate datetime

```

```

set @thisDate='6/15/' + cast(@thisYear as varchar(4))

insert into wellunpluggedfee (wellkey, duedate, amount)
select
w.pkey
, @thisDate
,100
from well w
where
        w.welltype NOT IN ('EOR','SWD')
        and w.fieldname <> 'WILDCAT'
        and w.wellstatus IN ('AA','CI','CPL','DG','I','NR','PR','WS')
        and w.pkey NOT IN (select wellkey
from wellunpluggedfee where wellkey =w.pkey and duedate = @thisDate)

```

Void Receipt Procedure

Database

```

x-- add IsVoid column to Receipt table (bit, default 0)

x-- Remove ReceiptKey column from Permit table (they are all null)

```

Data Access Layer

```

x-- Update Receipt object to get IsVoid

x-- Remove ReceiptKey from Permit

```

ReceiptList

```

x-- Add IsVoid column

```

Receipt Editor

```

x-- add read only checkbox for IsVoid

x-- change Delete to Void

x-- on void

x--- remove all receiptinstrumentallocations

x--- set isvoid = true

x- SetTitle not happening

```

Perforations vs. Formations

Perforation and formation data are managed in the same RBDMS.NET SQL Server table. Therefore, if a *ConstructPerforation* record includes a *FormationKey*, the

record refers to a formation. Otherwise, the record refers to a perforation event.

Inspector Search and Assignment

Within RBDMS.NET, the inspector search returns the following results:

- Agency staff: children of the MSOGB entity (determined by EntityName='MS OIL & GAS BOARD') with EntityType='STAFF'.
- Inspector: ANY Entity where EntityAddress.RoleCode='INSP'.

To assign an inspector to an approved permit, the following business logic is used:

```
Public Function SetInspector(ByVal prmt As Permit) As Int32?
    Dim insp As Int32? = Nothing
    Try
        Dim db As New Reference_DataClassesDataContext
        If prmt.FieldNumber.HasValue Then
            Dim qry = From c In db.RefFields Where c.FieldNumber =
prmt.FieldNumber Select c.Inspector
            insp = qry.FirstOrDefault
        End If
        If Not insp.HasValue AndAlso prmt.CountyNumber.HasValue Then
            Dim qry = From c In db.RefCounties Where c.CountyNo =
prmt.CountyNumber Select c.Inspector
            insp = qry.FirstOrDefault
        End If
        Catch ex As Exception
            insp = Nothing
            Throw New Exception("Permit_DataClassesDataContext.SetInspector: " &
ex.ToString, ex)
        End Try
        Return insp
    End Function
```

PermitApprove Right

A *PermitApprove* right is associated with the operations to approve, to void, and to place on hold applications for Form 2, Form 6, Form 18, and Form 19 permits. Any user that has been granted the *Read* permission of the *PermitApprove* right will be able to approve, void, or hold a permit application. Those without the right or without the read permission will not be able to perform those actions.

7520 Reports

The 7520 reports are based on the following stored procedures, which can be

found under the Programmability folder of the MSRBDM database in SQL Server Management Studio:

- `dbo.EPA7520_1`
- `dbo.EPA7520_2a`
- `dbo.EPA7520_2b`
- `dbo.EPA7520_3`

In addition, a number of table-valued functions are also used, and each is labeled consistently with the sections on the EPA 7520 printed forms:

- `dbo.GetEPA7520_1`
- `dbo.GetEPA7520_2A_V`
- `dbo.GetEPA7520_2A_VI`
- `dbo.GetEPA7520_2A_VII`
- `dbo.GetEPA7520_2B_V`
- `dbo.GetEPA7520_2B_VI`
- `dbo.GetEPA7520_2B_VII`
- `dbo.GetEPA7520_3_VA`
- `dbo.GetEPA7520_3_VB1`
- `dbo.GetEPA7520_3_VB3`
- `dbo.GetEPA7520_3_VB4`
- `dbo.GetEPA7520_3_VIA`
- `dbo.GetEPA7520_3_VIB`
- `dbo.GetEPA7520_3_VIC`
- `dbo.GetEPA7520_3_VID`
- `dbo.GetEPA7520_3_VII`
- `dbo.GetEPA7520_4`

System vs. user Dates

RBDMS.NET uses two varieties of *Date records that are differentiated by the value of the *IsSystemDate* column in the ***Date** table (1 = is a "system date," 0 = is not a system date). The test cases above have demonstrated that RBDMS.NET allows for adding, editing, and deleting user dates in each module as required. However, in addition, RBDMS.NET automatically writes system dates for events important in preserving well histories. Fields that are tracked for such changes in the database include members of the refCode groups named **DateTypes*. System dates are read-only in the user interface for all users by very strong developer recommendation.

Against developer recommendations, the MSOGB mandated that some means be left for a power user to edit a system date. Therefore, to comply with this direction, the developers have added a right in the security database that is referenced in RBDMS WinAdmin.NET as *SystemDateEdit*. Users who have this right can edit (but not add) RBDMS.NET system dates by *PKey* or **RefCodes.Code**.

Serious Warning: Editing *PKey* fields can lead to database-corrupting internal SQL Server errors if the edit causes a successive *PKey* to be incorrectly incremented to that same value. In other words, use of this right had better be reserved for people who know what they are doing.

Adding Reports

Available Report Formats

RBDMS.NET offers database administrators the option of two different types of report templates: SQL Server Reporting Services report language definition client (.rdlc) and extensible markup language (XML) grid-style reports.

RDLC Format

The .rdlc templates include formatting and layout options not available in the grid-style reports and can be developed either in Visual Studio or in the SQL Server report designer. The .rdlc format requires use of the ReportViewer control, and no preview is available without running the project. The .rdlc format is preferred for intranets and small Web sites rather than the report language definition (.rdl) format typically used with SQL Server Reporting Services installations because many agencies prefer not to run SQL Server Reporting Services. For Web sites with heavy usage, or if your reports make intensive use of subreports, you should consider using the .rdl format. The RBDMS Reports module can accommodate either format. However, since the .rdlc format has proven to be easier to use with RBDMS applications, this manual covers only the client-side processing of reports. It does not include the steps to run or preview .rdlc reports from within Visual Studio, which requires a separate window with a ReportViewer control to host the .rdlc report. RBDMS.NET 2.0 in Mississippi has been upgraded to use Version 10 of ReportViewer.

The difference between the .rdl and .rdlc formats is that, when the .rdlc format is passed to the report viewer, the <query> element of the Report Definition Language schema is not processed. The .rdl and .rdlc formats have the same XML schema. However, in .rdlc files, some values (such as query text) are allowed to be empty, which means that they are not immediately ready to be published to a Report Server. The missing values can be entered by opening the .rdlc file using the SQL Server 2008 version of Report Designer. (You have to rename .rdlc to .rdl first.) The .rdl files are fully compatible with the ReportViewer control runtime. However, .rdl files do not contain some information that the design-time of the ReportViewer control depends on for automatically generating data-binding code. By manually binding data, .rdl files can be used in the ReportViewer control.

Additional tips for creating .rdlc reports are presented in the other topics presented in this section.

Grid-style Reports

Please also see the help manual for the RBDMS WinAdmin.NET application for more information about grid-style reports.

Structure

The grid-style reports are derived from XML files and can be sorted and filtered with the column header controls and output as Adobe Acrobat (.pdf), Microsoft Excel (.xls), or Microsoft .xps files. An example of the structure of a grid-style report is shown below. The filter set that defines the selection criteria is associated with the report through the RBDMS WinAdmin.NET application's **Grid Reports Edit** form, which opens when the report name label on the tree control on the **Report** form is double-clicked.

	<code><?xml version="1.0" encoding="utf-8" ?></code>
A tag denoting the type of file...	<code>- <RBDMSReport></code>
The collection name that defines the report filter ...	<code><CollName>vwWeeklyReport</CollName></code>
A short description of the report...	<code><Description>Weekly Report</Description></code>
The sql SELECT command that defines the content of the report...	<code><sql>SELECT PermitDate, TypeDate, PermitStatus, Permit,PermitSubTypeI, Pool, PermitType, Elevation, ApplicationDepth, ApplicationNumber, EntityName, Address1, City, State, PostalCode, CommAddress, Lease, WellNumber, CountyName, Township, TownshipDir, RangeDir, Range, FootageNS, NS, FootageEW, EW, LandType, LandNumber, QuarterQuarterQuarter, QuarterQuarter, Quarter, OrientationType FROM vwWeeklyReport WHERE (TypeDate = 'Issued')</sql></code>
Close file type tag.	<code></RBDMSReport></code>

Display Options

You can change the look of the grid-style reports by invoking *Group by...* options. To change the default view, drag and drop a column header from the grid to the space labeled **Drag a column header here** to group by that column above. Then, to undo your *Group by...* selection and revert to the default view, drag the column header back to the data grid. You also can re-order the columns of data presented on the grids by clicking and dragging the column headers. For columns marked with a tiny down arrow , you also can customize your view of the data grid by selecting new column heads to display by choosing items from the column header dropdown lists marked.

Some agencies have found embedding a URL into a column of grid-style reports helpful for linking information views. This action is not directly supported in the RBDMS WinAdmin.NET application, but it can be done by editing the underlying XML. An example follows:

`<RBDMSReport>`

```

<CollName>Companies</CollName>
<Description>Active Well Operators</Description>
<sql>
  <![CDATA[
    SELECT EntityPKey, Operator,
      '<a href="http://somewebsite.com?urlparameter1=' + CAST (operator
as varchar(50)) + '" target="_blank">Click this link</a>',
    StatusDescription, Address1, City, State,
    PostalCode, Country, Phone
  FROM rptOperators
  WHERE 1=1
  ORDER BY Operator
  ]]>
</sql>
<Columns>Company
Filter<Column Name="EntityPKey" Width="80" Caption="Company
ID" /><Column Name="Operator" Width="150" Caption="Company
Name" /><Column Name="StatusDescription" Width="50" Caption="Status"
/><Column Name="Address1" Width="100" Caption="Street
Address" /><Column Name="City" Width="100" Caption="City" /><Column N
ame="State" Width="50" Caption="State" /><Column Name="PostalCode" Wi
dth="60" Caption="Zip" /><Column Name="Country" Width="80" /><Column
Name="Phone" Width="100" Caption="Phone" /></Columns>
</RBDMSReport>

```

Be sure to wrap the <sql> element in a CDATA tag and make sure that the **<a href>** information is correct.

Defining Reports

Reports are usually defined through several components:

- A data source, usually defined by a query (SELECT, WHERE, ORDER BY) run against a SQL Server database or a data view within SQL Server. Multiple table adapters that describe a set of these queries or views can be defined as a dataset within a Reports project in Visual Studio.NET 2008 and 2010. This dataset specifies the data source for purposes of report data binding. Visual Studio 2010 also gives developers the option of using an ADO.NET Entity Data Model (.edmx) as a binding source. However, examples in this manual assume the use of datasets in Visual Studio .NET 2010.
- A template with data fields bound to the fields exposed in the table adapter or view.
- A filter that contains elements compatible with the query or view and that is associated with the report parameters, the values of which may be defined programmatically by the host application or passed from one report to another.

The organization of templates for reports of similar type into a tree control for menu access is controlled through the RBDMS WinAdmin.NET application **Menus** section of the **Menus and Security** page. Once you have created a new report, you can add it to the target application **Reports** menu by adding a new node, setting the link to the file, and optionally specifying an icon file to display as an image on the tree control, all from the **Menus and Security** form in RBDMS WinAdmin.NET. Please see the RBDMS WinAdmin.NET help file for additional information.

As a convention, subreport templates names should be preceded with a lowercase "zz" prefix. With a "zz" prefix, the subreports will display after the top level reports to avoid confusion when browsing for reports.

Creating a Report Project

Visual Studio.NET 2008 and 2010 include an option to create a report project with a MainForm (2008) or a Form1 (2010) and Report1.rdlc items created for you. When you create a report project, many of the steps to start a report template are accomplished via a wizard. Although some people find the wizard easier to use, some options you may otherwise exercise are taken away and others are forced on you. Therefore, the following example will track the wizard action only through project setup and data source configuration.

To create a new report project, follow these steps:

1. Open Visual Studio.NET 2010 and click **File | New Project**. The **New Project** dialog box will open and allow you to target .NET Framework versions, choose templates, and name the project. Click **OK**.
2. The **Report Wizard** and the **Data Source Configuration Wizard** will both launch. In the **Choose a Data Source Type** dialog, choose Database. (**Note:** You *must* specify a data source for a report project because this wizard will launch automatically if the project does not contain any server connections or defined datasets. Other data sources besides SQL Server are available, but **only** SQL Server connections should be used in RBDMS.
3. For the database model, choose **Dataset** and click **Next**.
4. Choose the data connection or click **New Connection** to configure and test the connection.
5. Save the connection and choose your database object.

At this point, you may cancel the Report wizard and proceed manually.

Once the project is established, the process of creating a report can be accomplished in eight (compound) steps:

1. [Add the report](#) to the project.
2. [Specify the dataset](#) you will use to bind the report.
3. [Add header and footer](#) information
4. [Develop the report](#) body.
5. [Create report parameters](#) for filtering returned datasets and linking subreports.
6. [Create and associate filter controls](#) for your report.

7. [Add the report to the application Reports menu.](#)
8. [Test and debug](#) your report.

Adding a Report to the Project

1. In the Solution Explorer, right click the project name and, from the context menu, select **Add | New Item**.
2. In the **Add New Item** dialog, choose **Report**.
3. In the **Name:** field at the bottom of the dialog box, enter a name for the new .rdlc file.
4. Click **Add**.

Specifying the Dataset for Binding

1. In the **Report Data** pane, select **New | Dataset**.
2. In the **Dataset Properties** dialog, name the dataset, select the data source you will use to bind the report, and choose from the available datasets in the source.
3. Click **OK**.

Hint: Did you forget to include a field? Need to revise the query?

Sometimes while working with report projects, you will find that you need to edit the dataset or view underlying a particular report to add fields or to streamline a query. When that happens, follow these steps:

1. If your change is a minor one of adding or deleting a field from the dataset, open the **[datasetname].xsd** file by double-clicking it in the Solution Explorer, making your change in the table adapter, and saving the file. If your change is one of altering a view to add a table or significantly change the returned results, edit the view in SQL Server Management Studio and save your changes.
2. In Visual Studio, open the report you want to update with the new data source.
3. Refresh the data source for the report within Visual Studio.NET:
 - a. In the **Report Data** pane, right-click the data source and choose **Refresh**. OR
 - b. In the **Data Sources** pane, click the **Refresh**  icon.

Adding Header and Footer Information

1. From the **Report** menu, select **Page Header** to view the page header space.
2. Drag textboxes from the Toolbox into the header space and develop your

report header information, such as a title and other layout embellishments. Set any formatting desired in the **Properties** pane. Include placeholders for any report parameters you will be adding, such as *To:* and *From:* dates to filter the data to a specified range (see [Creating Report Parameters](#)).

3. From the **Report** menu, select **Page Footer** to view the page footer space.
4. Drag textboxes from the Toolbox into the footer space. To enter a run date or page numbers, right-click the empty textbox and click **Expression....** The **Expression** dialog will open. You can choose from and edit a number of useful expressions for building report information. Expressions are correct when the red squiggle underscores in the expression are completely cleared. Useful expressions for report footers include the run date `[=Now()]` and page numbers for printing `[="Page " & Globals.PageNumber & " of " & Globals.TotalPages]`.

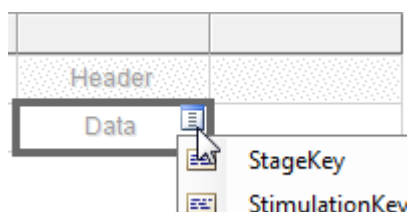
Hint: Headers and footers may be set to display on every page of the report, or be excluded from the first and/or last page. This option is a property of the Header/Footer which can be set in the **Header [Footer] Properties** dialog. Right-click in the header or the footer and choose **Header [Footer] Properties**. Footers should be used only when the report is long enough to span multiple pages.

Developing the Report Body

Laying Out the Report

Depending on your layout requirements, start by dragging a List box, a Table, or a Matrix control into the work area. Build the body of the report inside the list box or within the table or matrix control. List boxes and tables allow the report to "grow," so if you fail build the report with one of these container controls, the report will stop printing after the first record. While it is not impossible to add a list box after the template is in a state of advanced development, it is awkward and not fun.

You can add content to the report by dragging and dropping the desired fields from the list in **Report Data** pane into the table, matrix, or list control. Alternatively, you can accomplish this by clicking in each cell to select the desired field for binding from the shortcut menu accessed by the table icon, as shown below.



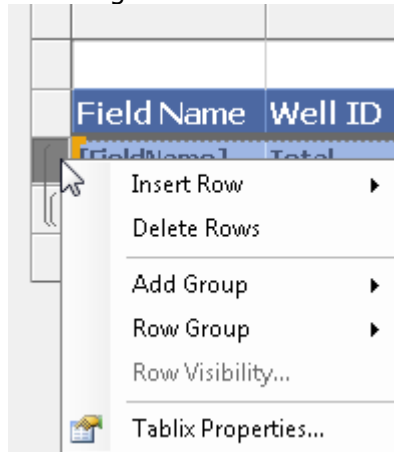
To change the configuration of the tablix or to add an expression, right-click in the container control.

You can add desired formatting to selected fields through the **Properties** pane. Use the multi-select keys (**Shift+click** and **Ctrl+click**) to select groups of fields to receive similar formatting.

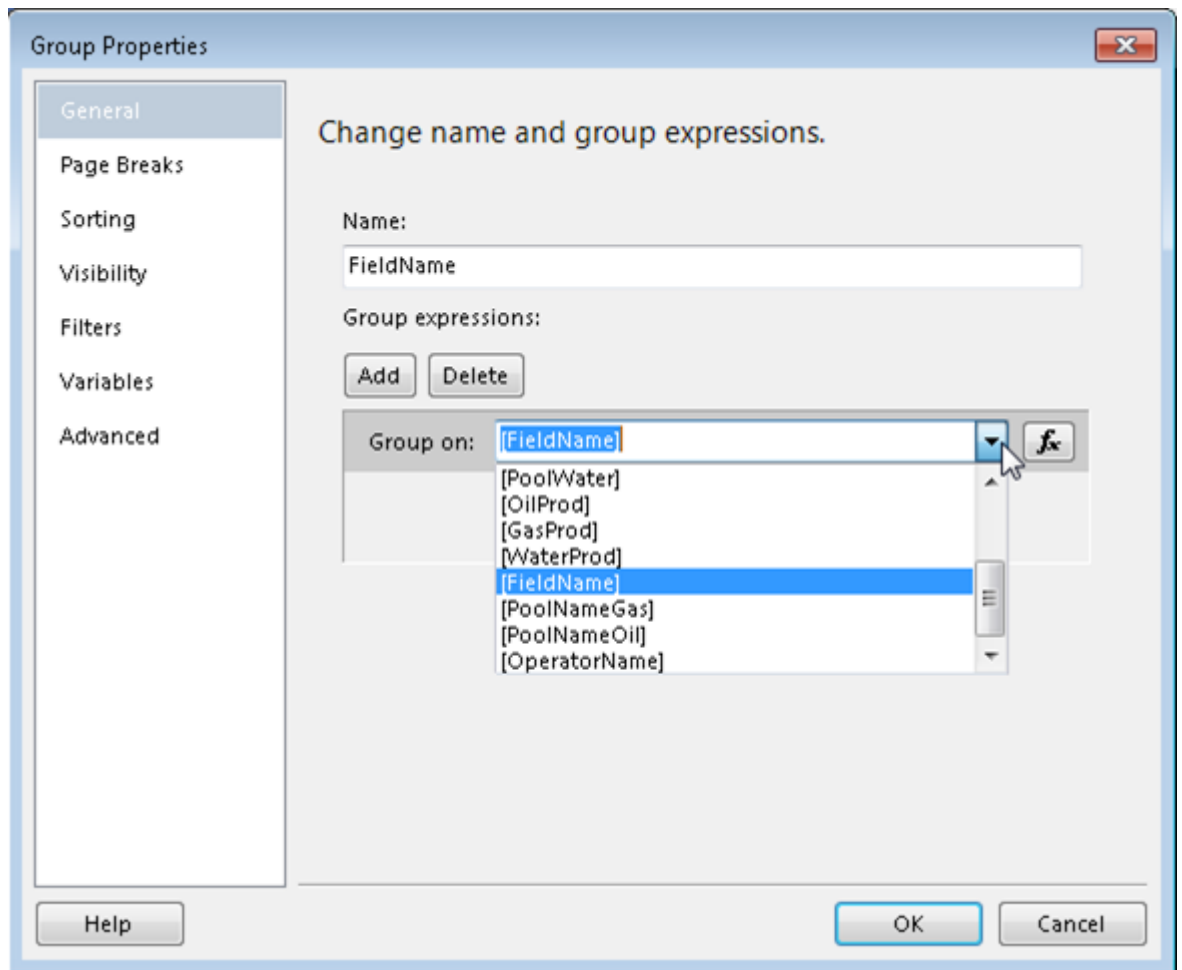
Grouping and Sorting

If you know that your report will have multiple groups and dynamic content, you might consider using the Report Wizard to specify the initial layout of the tablix, since a number of presets are available. But if your report will include tabular totals that require specifying a grouping and sort order and the available presets do not quite do the trick, follow these steps:

1. Right-click on the tablix row header where you would like to add a group for sorting. The shortcut menu allows a number of options for revising the tablix organization.



2. The **Add Group** selection will allow you to specify whether the group you are adding will be a parent group or a child group. To view the properties of an existing group, select **Row Group** from this shortcut menu. Be sure to specify the **Group on:** option in the **Group Properties** dialog.



3. To add a total to a grouped column, right click the tablix cell where you want the total to display and choose **Add Total | Column or Row**.

Adding Graphical Elements

For long grid reports, you may wish to introduce some eye-relief. To alternate background color on the report grid, use an "iif" expression for the BackgroundColor property for the row where you want the alternating color to begin, like so:

```
=iif(RowNumber(Nothing) Mod 2, "AliceBlue", "White")
```

To add an image to an .rdlc, follow these steps:

1. In the **Report Data** pane, right click the **Images** folder icon. Then click **Add Image** and navigate to the location of the image file.
2. Click **Open** to add the image to the project **Image** folder.
3. Drag the image to the desired position on the work area. The **Images Properties** dialog will open.
4. Specify the name of the image, the wording for a tooltip, if desired, and other properties associated with sizing, visibility, hyperlink action, and borders.

5. Click **OK**.

Hint: Although it is also possible to create a link to an external image to be used by the report or to store the image in the database, additional programming is required and the *EnableExternalImages* property must be set to *True* in ReportViewer. Therefore, using the **Embedded** option for the image source is recommended.

Fixing Undesirable Page Breaks

The reality is that an .rdlc template is basically a text file. Controlling the placement of elements is often a matter of squish, jiggle, check, re-squish, jiggle again, recheck, and cuss. However, here are two tips that may save you some time.

Horizontal Page Breaks

If you find that you are having problems with the layout stretching horizontally off your targeted paper size for printing, make sure you set the **CanGrow** option on the individual control properties on your report to *False* (unless that is desirable behavior).

Vertical Page Breaks

Sometimes when the body of a report contains tables and rectangles, rendering the page into a .pdf can result in large chunks of empty white space appearing on the first page and an unnecessary page break for the continued elements. The solution to this misery lies in tricking the .pdf export to recalculate the correct body size, like so:

In the **Report | Properties | Layout**, set

- Page width: 8.27in
- Page height: 11.69in

Use inches for this, since the .pdf export does not render centimeters correctly. Then set the margins like so:

- Left margin: 1.5cm
- Right margin: 0.5cm
- Top margin: 1cm
- Bottom margin: 1 cm

Finally, in **Body | Properties | Size**, delete entry. The body size will be recalculated.

Creating Report Parameters

1. In the **Report Data** pane menu, right click the **Parameters** folder and select **Add Parameter**. The **Report Parameters Properties** dialog box will open.
2. Define the parameter. Generally, it will be one of three types:
 - If the parameter will be used to filter the returned data to a specified range (*To/From date range, Operator, Well Type*, etc.) in the report, the name should be meaningful enough to link the

parameter name to the desired filtering function. You will need to build a filter control with the same name in RBDMS WinAdmin.NET. Click **OK**.

- If you are creating the report parameter for linking a subreport, the name should be the same as the field that will link the subreport to the main report. Be sure to select the correct data type and indicate whether a blank value is allowed (True). Click **OK**.
- If the parameter will be used to call a filter set, name the parameter *CollName* of data type *Text*, and enter a default value that will match the name of the filter set you will build in RBDMS WinAdmin.NET. Click **OK**.

An RBDMS Convention: The RBDMS.NET developers use the report parameter *CollName* to specify the name of the filter set created on the **Edit Filters** page in WinAdmin.NET. A report parameter *CollName* should therefore be added to each .rdlc file you plan to add to an RBDMS application menu. The **Default Value** you specify for the *CollName* parameter in each report should match the name of the filter set you define in RBDMS WinAdmin.NET for use with the report. The *CollName* parameter therefore serves to associate the report with the desired filter control collection.

Any other report parameter that you define to be passed to the .rdlc (e.g., start and end dates, etc.) also must be defined as a filter control in RBDMS WinAdmin.NET and assigned to the *CollName (NameYouChose)* filter set on the **Edit Filters** page of RBDMS WinAdmin.NET.

You do not have to define a parameter for the filterable columns because those are appended to the WHERE clause of the sql unless the *ParameterOnly* option in the filter control defined in RBDMS WinAdmin.NET is set to *True*. In this case, the column does not get added to the WHERE clause and is passed as a report parameter.

3. Drag the report parameter from the **Report Data** pane into a textbox control in the correct position in the report. For example, in the case of a parameter that will be used to filter data return to a specified *StartDate*, the correct position probably will be in the report header. For a parameter that will be used to link a subreport, the correct position will be some discrete location in the body of the subreport, and the visibility of the textbox control containing it should be set to *Hide*.

Creating and Associating Filters

The dynamic filtering parameters associated with each report are defined in RBDMS WinAdmin.NET on the **Edit Filters** form. These filter controls give you the ability to limit the results returned by the query underlying the report to the dataset of interest. The WinForm version of the application can be downloaded from <http://RbdmsOnline.org/downloads/MS/RbdmsWinSetup.zip>.

From the **Start** menu, select **All Programs | GWPC | RBDMS WinAdmin**.

1. In RBDMS WinAdmin.NET, make sure that your database connection is pointed to your correct instance of the database on the **Forms | Configuration Options** page, **Database Connection** tab. If it is not already pointing to the correct installation, reset the connections.
2. With Windows Explorer, copy the .rdlc file you created into the folder shown for the appropriate report path, as shown in **Configuration | Settings** page.
3. In RBDMS WinAdmin.NET, select **Forms | Edit Filters** to open the two-paned page that is used to manage report filter controls.
4. In the left pane on the **New Record** row, enter the *CollName* you specified as a report parameter in Visual Studio.NET. Enter a description for the filter collection in the **Description** column. this will be your *Set Name*.
5. Build the filter collection to include those needed to handle any report parameters that your users will need to retrieve information. To do this, select the desired filter controls on the right and then drag and drop them on the **Set Name** record you created. You may need to build the necessary individual filter controls in the right pane first by completing the form grid. Be sure to test any .sql statements included in individual filter controls in SQL Server Management Studio before using them.
6. Right click the **Set Name** record to preview the filter.

Hint: You may delete items from your set on the left pane without deleting the control from the database. However, deleting filter controls from the right pane is a permanent action.

If you must create a new filter control on the right pane of the **Edit Filters** page in RBDMS WinAdmin.NET, the following data dictionary snippet may provide useful hints in how to complete each new control record. The snippet shows the columns in the DevControls table, which governs the filtering in RBDMS.NET:

Column Name	Data Type	Length	Nulls	Default	Description
ControlName	varchar	50	NO		Unique name for the control (i.e., "Operator")
xml	text	2147483647	YES		Contains the XML serialization of extra information for the control object
ColumnName	varchar	50	YES		Column name for this control. Can include table name if needed.
Prompt	varchar	50	YES		The label to create next to the left of the comparison and criteria controls.
DataType	varchar	255	YES		The data type for the criteria ("String","Integer","DateTime","Single","Double")

Column Name	Data Type	Length	Nulls	Default	Description
ControlType	varchar	255	YES		The control type to use ("TextBox","ComboBox","CheckBox","DateCombo","ListBox")
DefaultValue	varchar	255	YES		A default value to use for the criteria.
Tooltip	varchar	255	YES		A tooltip to show when the cursor is over the control. Useful when you need a longer description than will fit in the label area.
CompareItems	varchar	255	YES		A semicolon delimited list of SQL comparison operators to make available (<; <=;; >; >=;; <>; Like)
DefaultCompare	varchar	10	YES		The default comparison criteria (i.e., "=").
HideCompare	bit	1	YES	((0))	If true then the comparison control will be hidden. This is helpful if you are using the default value for the comparison control and you don't want the user to change it.
InputMask	varchar	50	YES		A data input mask to use for a textbox control.
MinValue	varchar	50	YES		The minimum allowable value.
MaxValue	varchar	50	YES		The maximum allowable value.
SQL	varchar	1024	YES		A SQL statement to retrieve data to display in a combo or listbox.
DisplayField	smallint	2	YES		The integer position of the description field in the SQL query result (0 based).
ValueField	smallint	2	YES		The integer position of the value field in the SQL query result (0 based).
ListItems	text	2147483647	YES		A semicolon delimited list of items to use in a combo or listbox. This is an alternative to populating using a SQL query.
Hidden	bit	1	YES		True if the control is hidden and always used with a defaultValue and DefaultCompare
Format	varchar	50	YES		Format specification
Sort	bit	1	YES	((0))	If true then this field will be available for sorting.
Required	bit	1	YES	((0))	If true this filter criteria must include non-null comparison criteria and comparison value.
ParameterOnly	bit	1	YES	((0))	If true this criteria will be passed as a parameter only and not part of the WHERE clause.

Adding Reports to the Application Menu

Once the report is built, you can add the report to the application through RBDMS WinAdmin.NET for the smoke test. Menu management is done in RBDMS WinAdmin.NET.

1. Open RBDMS WinAdmin.NET, switch to the **Security** tab of the **Menus and Security** page.
2. In the **Rights** pane, scroll to the **New Record** row and specify the right for the report you wish to add.
3. Expand the **Rights** record and assign the *Everyone* role to the right. Be sure to click off the new record to commit the write action.
4. Switch to the **Menus** tab of the **Menus and Security** page.
5. In the **Select Menu** dropdown, select *Reports*. This menu is the live menu for the RBDMS.NET WPF **Reports** menu.
6. Add the report .rdlc to the menu by selecting the record in the right pane and dragging it to the desired node on the tree control in the left pane.
7. Click **Save Updates** to save the change to the menu.

Testing and Debugging Reports, Filters, and Menus

Reports are best tested and previewed within the target application in a test/development environment.

1. Open the target RBDMS application and select the **Reports** explorer.
2. Expand the **Reports** menu to confirm that your new menu item is visible and that, when invoked, each control in your defined filter set works as planned.
3. Check the report at runtime to determine any necessary formatting revisions or debugging.

A Word about Subreports

Subreports must be developed as separate .rdlcs from the main report and should not have a header or footer. A main report can contain many subreports. Subreports also may contain their own subreports if desired, but these must be defined within those subreports (not the main report). As a convention of the RBDMS developers, subreports are always named with the prefix "zz."

You can embed a sub-report into the main report body by selecting the **Subreport** control from the Toolbox pane in Visual Studio.NET. When you insert a subreport control on the main report body, you must specify the data link to the subreport as a named parameter. For example, in MSRBDMS.NET, the subreport **zzConstructCasing** is linked to the **Well Completions** report by the

ConstructKey which is specified as the parameter value *ConstructKey* in the **Subreport Properties** dialog.

Also in the **Subreport Properties** dialog, your entry in the **Parameters | Name** field of the linked subreport definition must match the parameter value name (in this example, *ConstructKey* must be added in the **Parameters | Name** field). If you forget to link the subreport properly to the main report, the subreport will not work.

Integration with Other Applications

Links to eReport.NET

Accepting monthly reports through the eReport.NET **Admin** page moves the data that has passed four levels of data integrity checks (client-side, schema, and server-side, and administrative server-side validations) into the RBDMS.NET production tables. Once the data is imported into RBDMS.NET, it is available for viewing and reporting purposes within the front-end application.

Public method calls for the **RbdmsServiceMS**, a Windows Communication Foundation (WCF) service, are documented in the eReport.NET administrator's help file.

Links to LaserFiche

MSRBDMS.NET has been linked to the MSOGB's LaserFiche documents repository through a Windows Communication Foundation (WCF) service named **RbdmsServiceLaserFiche**. When the **Documents** tab on the **Well Information** page of RBDMS.NET application is opened, code behind the form calls a web service *Get* method that relays the list of documents in the repository for the referenced API number to the user interface. This and the other service methods available to the **RbdmsServiceLaserFiche** service on the MS EXWEB server are listed below, as drawn from developer's comments in the code.

```
''' <summary>
    ''' Get the version of this instance of the web service. Useful for initial
    testing of connection to service.
    ''' </summary>
    ''' <returns>The service version.</returns>

    Public Function Version() As String Implements IServiceLF.Version
    ''' <summary>
    ''' Gets documents recursively starting from a root path passed to the
    service.
    ''' </summary>
    ''' <param name="dirPath">The dir path to start the search.</param>
    ''' <returns>A list of docItems</returns>

    Public Function GetDocumentPaths(ByVal tokenID As Guid,
    ByVal dirPath As String) As System.Collections.Generic.List(Of DocItem)
    Implements
```

IServiceLF.GetDocumentPaths

```
''' <summary>
    ''' Finds all documents that match the name parameter. If no "*"
    wildcards are included in the name then wildcards will be added to the
    start and end of the name.
    ''' For example if the search name="1212312345" then the actual
    search will be based on name="*1212312345*". If the name includes an
    asterisk then none will be added.
    ''' </summary>
    ''' <param name="TokenID">The token ID.</param>
    ''' <param name="name">The name.</param>
    ''' <returns></returns>
```

Public Function GetDocumentByName(**ByVal** TokenID **As** System.Guid,
ByVal name **As String**) **As** System.Collections.Generic.List(Of DocItem)
Implements IServiceLF.GetDocumentByName

```
''' <summary>
    ''' Gets a document (returned in a byte array in PNG format) by passing
    the path to the document.
    ''' </summary>
    ''' <param name="tokenID">The token ID.</param>
    ''' <param name="filePath">The file (document) path.</param>
    ''' <returns></returns>
```

Public Function GetDocument(**ByVal** tokenID **As** Guid, **ByVal** filePath **As**
String) **As Byte()** **Implements** IServiceLF.GetDocument

Links to eForm

A separate help manual has been provided for the eForm Web application installed on EXWEB. The eForm workflow has been configured to flow into the RBDMS Permit module once all data validation checks have been passed. The ePermitWindows service is the component that moves the data into the internal application and pushes permit status changes back to the main navigation page of eForms from the RBDMS.NET application.

Links to Log Images

The association of log images with wells is being built with the 10-digit API number. Once the information is in the table, the WellImage is linked to Well via **Well.PKey=WellImage.WellPKey**.

The developers added three additional columns to the **WellImage** table to accommodate log files: **ImageType**, **ImageSubType**, and **SeqNo**. The content of these fields indicate how the file names are being parsed. The **ImageType** must be *SCOUTCARD* or *LOG*. (note capital letters and no spaces). The **ImageSubType** is the code now in use to designate the kind of log. **SeqNo** is the trailing digit in the file name that MSOGB has been using to

designate the next log of the same type.

The parsing of the file names ignores underscores, spaces, and the trailing file extension. Inconsistently named files in the catalog containing these images will not be available for viewing.

Microsoft Office CiceroUIWndFrame

Microsoft Office 2003 has a known issue with a component calls CiceroUIWndFrame, which controls alternate user input components such as handwriting and voice recognition. If Microsoft Office components become unstable on a workstation, RBDMS.NET also may crash and display an error message that says "**There is no disk in the drive. Please insert a disk into drive _:**".

This error can be dealt with in two ways if it occurs. The first is a work-around:

1. Put a CD in the drive.
2. Start RBDMS.NET and log in.
3. Take the CD out of the drive.
4. Exit RBDMS.NET.
5. Start RBDMS.NET and log in, which will trip the error.
6. Then put a CD back in the drive and click **Try Again**. The program will start without the error.

If the Alternative User Input modules in Microsoft Office are not needed, they can be removed, which should stop the CiceroUIWndFrame error:

1. Run Office's setup routine from **Add/Remove Programs** and look in the list of installed Office programs.
2. Select **Office Shared Features | Alternative User Input | Speech and Handwriting**.
3. Disable **Speech and Handwriting**.

Managing RBDMS.NET Security

Using RBDMS WinAdmin.NET

The RBDMS.NET Windows Administrative (WinAdmin.NET) application was created for use by developers and database administrators to manage multiple Internet- and intranet-based applications, such as RBDMS, Data Mining, eReport.NET, and eForm (ePermit). Users who are not members of the administrator or developer role should not be given access to the RBDMS WinAdmin.NET application because its misuse could result in application malfunction, data loss, or both. The WinAdmin.NET application is available as a desktop smart client application and controls the following functions of MS RBDMS.NET:

- Application configuration
- Security (users, roles, and rights)
- Menu management
- Filtering and reporting
- Data validation

Please see the RBDMS WinAdmin.NET user's manual for help in managing these and other functions.

Managing the Production Environment

Before entering production mode initially, the MSOGB database administrator should set up an *RBDMS* role in SQL Server with the appropriate rights and include the MSOGB users in the role. Rather than adding each user as *db_owner*, the developer suggests adding the built-in roles of *db_datareader* and *db_datawriter* to each user. This configuration should allow read/write permissions to all the tables but not allow design changes, but this configuration also should be tested first. Grant execute permissions to the user with the following syntax:

```
USE <dbname>
```

```
GRANT EXECUTE TO <username>
```

You need to grant execute on **both** RbdmsSecurity and MSRBDMS databases.

Please consult the Microsoft SQL Server for additional guidance.

A copy of the live front-end application and back-end SQL Server database should be preserved in a secure location and backed up. Future updates, additions, and changes should be tested in a separate development environment with a copy of the MSRBDMS.NET. After testing is deemed successful, the revisions should be copied to the server location for availability to the MSOGB staff in accordance with the MSOGB deployment strategy.

Backing Up, Restoring, and Maintaining the RBDMS.NET Database

The system administrator should have an adequate backup plan in place to protect from potential disasters such as theft, fire, flood, hardware failure, and operator error. Other maintenance requirements include periodic duties such as adding users, groups, and security as needed, monitoring database and log space

usage, and increasing available storage space when needed.

Specific backup procedures will depend on factors such as the type of database and frequency of updates. As a minimum, the database should be backed up daily to a removable medium. A system for rotating backup media so that a copy is stored offsite also is highly recommended. Removable media stored onsite should be kept in a fireproof safe.

You also should check the task history periodically to ensure that the backups are being performed properly. For example, in SQL Server, the transaction logs are usually truncated at each checkpoint. If you wish to apply the transaction logs after restoring the system to recover additional data, then turn off the **Truncate after Checkpoint** option and have the transaction log dumped after the backup each evening.

For SQL Server maintenance, you should schedule two other tasks each evening in addition to the backup. You should run a "**dbcc checkdb(database name)**" command to check and repair any problems with the database. You also should run a stored procedure to update table statistics to improve the ability of SQL Server to optimize queries. The administrator should check the history for these tasks periodically to ensure that they are operating correctly. Please consult the Microsoft SQL Server help system for guidance in setting up these tasks.

The two primary reasons for performing a restore of the database are hardware failure that makes the disk unreadable or critical operator error. If that happens, you will need to repair any hardware as needed and then restore the last database backup. Keep in mind, however, all data changes since the previous backup will be lost.

Steps to restore the RBDMS.NET SQL Server 2008 database are as follows:

1. Open SQL Server Management Studio on the server and expand the list of databases in the Object Explorer.
2. Right-click the MSRBDMS database and, from the context menu, choose **Tasks | Restore | Database....**The **Restore Database** dialog will open.
3. On the **General** page of the **Restore Database** dialog, select or type the name of the database for the restore operation, in this case *MSRBDMS*, and click **From device**. Click the **Builder**  button to navigate to the location of the backup sets to restore.
4. In the **Specify Backup** dialog, select *File* as the **Backup Media** and then click **Add** to drill to the location of the backup file (**MSRBDMS.bak**). Click **OK**.
5. Click OK to close the **Specify Backup** dialog.
6. Click the Restore check box next to the database name in the **Select the backups sets to restore:** pane.
7. Switch to the **Options** page of the **Restore Database** dialog.
8. For the **Restore options**, click **Overwrite the existing database**.
9. In the **Restore the database files as:** pane, specify the target locations for the **RBDMS_Data**, **RBDMS_Log**, and **sysft_MSRBDMS_FTS** files.
10. For the **Recovery state**, click (RESTORE WITH RECOVERY).

11. Click **OK**.

Please see the extensive Microsoft SQL Server help system for more information.

Replicating the Database

Setting up merge replication between the field inspection laptops and the MSRBDMs SQL Server is an extremely complex task. Unless you are very proficient with SQL Server administration, the developers recommend that you get expert help before attempting this procedure. Whole books are dedicated to this aspect of SQL Server administration, and the Microsoft help system on this topic is both extensive and daunting. However, to acknowledge that every database administrator in state agencies that use laptops in their field inspection programs must face this task, a brief synopsis of the general procedure for setting up merge replication is presented here.

1. Install the .NET Framework 4.0 on each laptop.
2. Install SQL Server 2008 Express Edition with Advanced Services on each laptop. This version includes SQL Server Management Studio Express, SQL Server Reporting Services, and the Full-Text Search engine.
3. Create a merge replication publication on the production SQL Server 2008 for the MSRBDMs database (i.e., all tables, all columns and all rows). This was done in late September 2009 in SQL Server 2005 as **MSRBDMs200907_Merge** on the MSOGB RBDMS-NET server.
 - a. Ensure that SQL Server is installed and running.
 - b. Check disk space. More than 1 GB is recommended
 - c. Ensure that the SQL Server agent is running and that the service is running under a login that has access rights to the **MSRBDMs200907_Merge** publication on the (RBDMS-NET) server running the production version of RBDMS. **Note:** This should **not** be a user account because, if the password changes, the service will fail.
 - d. Open Management Studio and make sure that the local SQL Server is registered under the computer name and not "(local)."
 - e. Right-click **Replication** and choose **New | Subscriptions**. The New Subscription Wizard will open.
 - Choose Publication <servername>\MSRBDMs (**RBDMS-NET\MSRBDMs200907_Merge**).
 - Choose **Destination database**. Create new "MSRBDMs" database.
 - Choose to run snapshot agent immediately.
 - Click **Finish**. The first replication will take about 5 minutes.

Configuring Source Code

RbdmsDAL is the data access layer implementation for RBDMS.NET One of the prime directives of the RBDMS program is to maintain a "core" set of functionality

while allowing implementation-specific customization. Ideally, "core" functionality would reside in compiled assemblies to prevent deviation. In practice that is impossible at this time. To prevent deviation from the RBDMS "core" the following provisions and rules have been made for implementation-specific customization:

1. DO NOT EDIT ANY .DBML FILE!!!

The classes modeled by the DBML represent the "core" data access and **MAY BE REPLACED AT ANY TIME!** Instructions for the foreseen use-cases of adding database fields and implementing business logic are as follows.

2. Adding Database Fields

Each **[Name]_DataClasses.DBML** file has a corresponding **[Name]_Custom.vb** file allowing addition of database fields. An example is the Well entity. The source code includes a **Well_DataClasses.dbml** and a **Well_Custom.vb**. The **Well_Custom.vb** file is pre-filled with partial classes for the datacontext and each entity defined in the **Well_DataClasses.dbml** file. To add a database field, you would add code to the appropriate entity class in **Well_Custom.vb**. The following VB.NET code snippet reflects the most basic case of how to start. Typing the characters addcol invokes the auto-completion. Please consult the LINQ to SQL documentation for more advanced scenarios.

```
Partial Private Sub On$PropertyName$Changing(ByVal value As
$PropertyType$)
End Sub
Partial Private Sub On$PropertyName$Changed()
End Sub
Private m_$PropertyName$ As $PropertyType$
<Column(Storage:="m_$PropertyName$", DbType:="$DBType$")> _
Public Property $PropertyName$() As $PropertyType$
    Get
        Return Me.m_$PropertyName$
    End Get
    Set(ByVal value As $PropertyType$)
        If ($PropertyType$.Equals(Me.m_$PropertyName$, value) =
False) Then
            Me.On$PropertyName$Changing(value)
            Me.SendPropertyChanging()
            Me.m_$PropertyName$ = value
            Me.SendPropertyChanged("$PropertyName$")
            Me.On$PropertyName$Changed()
        End If
    End Set
End Property
```

3. Adding Business Logic

Each **[Name]_DataClasses.dbml** file has a corresponding **[Name]_BLL.vb** file allowing addition of business logic. Again, for example, for the Well entity, the source code includes a **Well_DataClasses.dbml** and a **Well_BLL.vb**. The **Well_BLL.vb** file is pre-filled with partial classes for the datacontext and each entity defined in the **Well_DataClasses.dbml** file. Each partial class is already coded to inherit from **RbdmsBasBusiness.BaseBusiness** to facilitate further customization and future enhancements. To add business logic you would add code to the appropriate entity class in **Well_BLL.vb**, as follows:

Partial Class WellAKA

Inherits RbdmsBaseBusiness.BaseBusiness

Private Sub WellAKA_PropertyChanged(ByVal sender As Object, ByVal e As System.ComponentModel.PropertyChangedEventArgs) Handles Me.PropertyChanged

Me.IsDirty = True

ExecuteValidationChecks()

End Sub

Private Sub ExecuteValidationChecks()

Me.CheckAlias(Me.Alias)

Me.CheckAliasType(Me.AliasType)

End Sub

Private Sub CheckAliasType(ByVal value As String)

If String.IsNullOrEmpty(value) Then

Me.AddError("AliasType", "Alias type is required.")

Else

Me.RemoveError("AliasType")

End If

End Sub

Private Sub CheckAlias(ByVal value As String)

If String.IsNullOrEmpty(value) Then

Me.AddError("Alias", "Alias is required.")

Else

Me.RemoveError("Alias")

End If

End Sub

End Class

Working with the Dynamic User Interface

About the Dynamic User Interface

Dynamic User Interface (DynUI) refers to a technique of generating database data entry/view screen forms from a combination of external XML files and the schema of the underlying database. The benefits of this technique are multi-layered. One immediate effect is that specifying the forms becomes much easier by following the database schema flow. The technique also enforces a common look and feel throughout the RBDMS family of applications by restricting the user interface to standard (within RBDMS), well-defined elements. Using the DynUI approach also makes it easier for the developer to modify the screens by simply changing the external XML files without recompiling the application, increasing his or her productivity. When the application development is complete, the files can be converted into resources so that they cannot be changed or modified by the end user.

Changing the UI without recompiling may prove to be useful over the lifetime of the RBDMS application in routine matters such as changing the order of fields, hiding or showing fields, or changing their characteristics (for example, making a field read-only).

Hiding an entire screen is possible just by removing the corresponding DynUI XML file from the application or by setting a property in the XML file. However, adding a new screen is not normally possible just by changing the XML, since the screens correspond directly to the database schema as visually designed in the Visual Studio.NET Object Relational Designer (O/R Designer, the DBML designer).

DynUI File Folder

The DynUI XML files are contained in the **DynamicUIXml** folder underneath the directory containing the executable RBDMSWPF.EXE file. If the DynUI XML are saved as *Resources* instead of *Content*, then they do not appear in the **DynamicUIXml** folder.

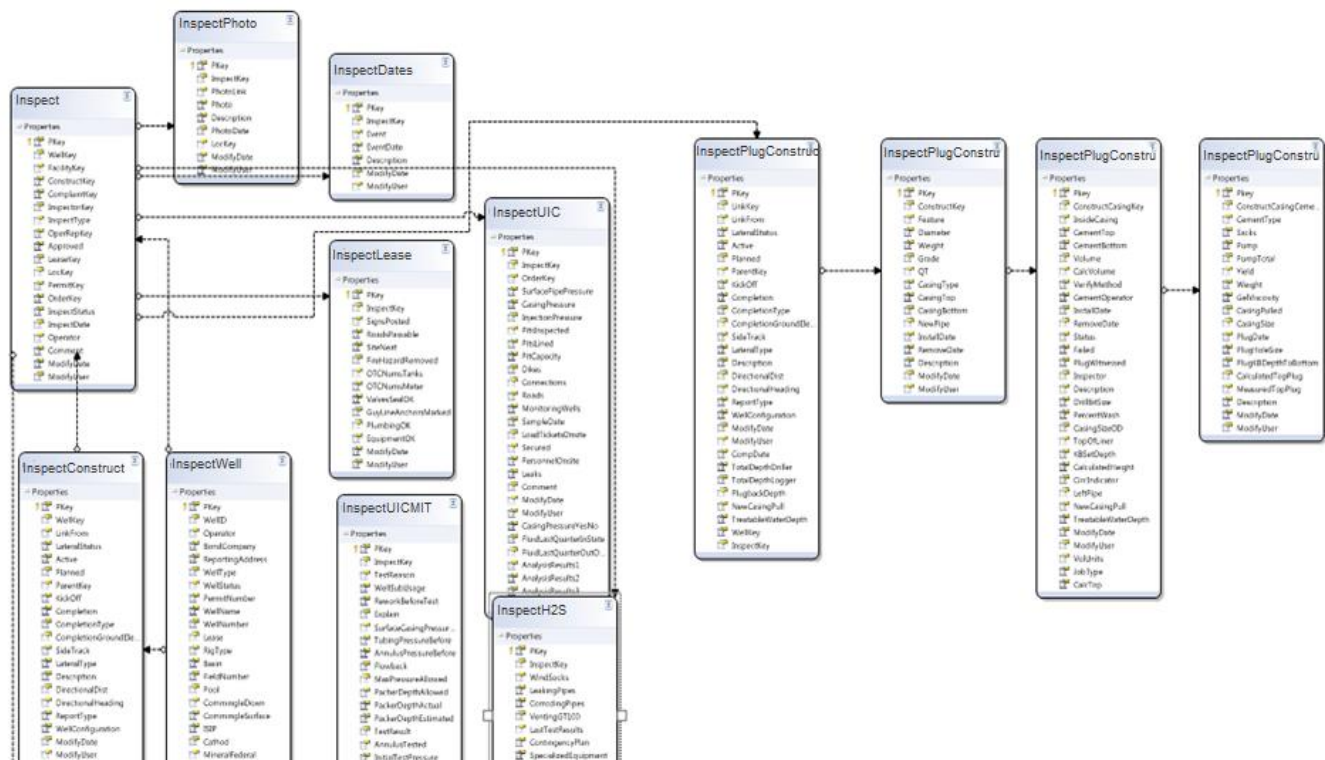
DynUI XML Files

Within the DynamicUIXml folder are a number of XML files called xxxxxx_UI.XML, where xxxxxx is the name of the table (Linq class) being displayed, with an -s added (pluralized). For example, the table **IncidentVolume** would have a corresponding **IncidentVolumes_UI.XML** DynUI file.

General Description of the RBDMS.NET User Interface

The RBDMS.NET UI was designed to mirror the database structure as expressed in the Visual Studio.NET Object Relational (O/R) Designer. In other words, the O/R designer is used to show tables and the relationships between tables (one-to-many, one-to-one, etc). The DynUI then displays the data on the screen using these tables and relationships. The O/R designer can represent not just tables, but any LINQ (Language Integrated Query) class. For example, you might have several different ways to look at an Entity table, such as Officer or Branch Office. Even though they use the same underlying table (Entity), they can have their own LINQ class in the Visual Studio O/R designer. Each LINQ class behaves like an independent table in the Dynamic UI. This means that in the following

descriptions, anywhere a table is described, this could be any LINQ class. Here is a sample view of the O/R designer for the Oklahoma RDBMS.NET Inspect module:



Each main screen in RBMDS starts with a table (LINQ class); for example, Entity, Well, or Inspect. Related subtables are shown as tabs below the main table. In the image below, Inspect is the main table with subtables InspectDates, InspectLease, InspectSWD/UIC, InspectMIT, InspectH2S, and InspectPhotos. Each of these tabs can then have subtabs, if they have subtables.

The following screen capture shows the Inspection main table, with the MIT tab showing, with three MIT inspections. This shows at a glance what MIT inspections have been performed.

Inspection Information

Well Name: EXXON WALNE
 Well API: 35-009-20825
 Incident:
 Inspect Type:
 Target:
 Inspector:
 Comment:

Operator: CHESAPEAKE OPERATING INC
 Status:
 Inspect Date: 4/20/2010
 Operator Rep:

Apply Changes Delete

Test Type	Well Type	Pass / Fail	Last modified by	Last modified on
5 Year Periodic	Gas	Pass	tester	4/20/2010 12:40:48 PM
Division Request	Gas	Pass	tester	4/20/2010 12:41:03 PM
Operational	Gas	Fail	tester	4/20/2010 12:41:18 PM

If you then press the **Builder** button, the detail form for the MIT inspection will then display:

Inspection Information

Test Type: Division Request
 Well Type: Gas
 Pass / Fail: Pass
 Last modified by: tester
 Last modified on: 4/20/2010 12:41:03 PM

Rework Before Test? ☐

Explain

Surf Casing Press., Before Test: 23
 Annulus Press. Before: 44
 Max. Press. Per OCC Order: 100
 Actual Packer Depth:
 STARTING TEST PRESS.: 99
 Positive Annulus Pressure Monitoring? ☒
 WELL SHUT DOWN? ☐
 Alternative Testing Procedure Authorized by OCC Order No.
 Tracer Survey Passed? ☒
 Tubing Pressure Passed? ☒

Tubing Press., Before Test: 44
 Flowback Volume: 0
 Packer Depth Per OCC Order:
 Estimated Packer Depth:
 ENDING TEST PRESS.: 99
 Reason:
 Fluid Level Passed? ☒
 Fluid Depression Passed? ☒

This gives the RBDMS.NET user the ability to drill down into more detail whenever required.

This example shows that, in general, the fields in a table can be displayed in either a form view or in a tabular view (list view). The DynUI XML files drive these two views.

About the DynUI XML Structure

The DynUI XML is organized into two main sections, a `<RBDMS_UI_FormView_Description>` section, and a `<RBDMS_UI_ListView_Description>`. The [Form View](#) section instructs the application how to display the table in a form, and the [List View](#), how to display the table in a list.

Sometimes a table will have just a FormView, or just a ListView, or both. For example, the main table (e.g., Entity, Well, Inspect), will normally only have a FormView and not a ListView. Subtables usually have both, although sometimes only a ListView, if no form expansion is needed.

[The element](#) `<RBDMS_UI_Description>` contains attributes that control the overall functionality of a table-subtable view.

RBDMS_UI_Description

SubTabOrder: Space-delimited String

- Names of EntitySets in order or desired appearance in tabs. Excluded names are included after the included names. Used to prioritize certain tabs.

This is used to change the order of the tabs of the sub-tables. For example:

```
<RBDMS_UI_Description Name="WellEditor" SubTabOrder="EntityOfficers  
Communications EntityAddresses  
EntityAffiliates" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/p  
resentation">
```

will have these tabs:



while

```
<RBDMS_UI_Description Name="EntityEditor" SubTabOrder="Addresses  
Communications EntityAddresses EntityOfficers  
EntityAffiliates" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/p  
resentation">
```

will have these tabs:



OneToOneRelationsToDisplay: Comma-delimited String

- Names of entity types that are on the one side of a one-to-many

relationship with the Entity for which this UI Description is used that will be displayed. (Normally, the one side is not displayed).

This allows the designer to specify a LINQ entity that can be referred in an .xml file that controls the many side of a one-to-many relationship or a one-to-one relationship. Without this, you cannot refer to fields on the one side of a one-to-many relationship if the XML file is displaying the many side. For example, if you are displaying Construct records for a well, you can't refer to any of the fields in the Well record in the **Constructs_UI.xml** file. For example, if you want to show the Well.LeaseName field in the Constructs Form View, you would add the following:

```
<RBDMS_UI_Description Name="WellConstruct" Label="Construction
Records"
OneToOneRelationsToDisplay="Well" xmlns="http://schemas.microso
ft.com/winfx/2006/xaml/presentation">
```

Then in FormFieldDescriptions,

```
<FormFieldDescription Label="Lease:" Type="Label" Row="6" Column="3"
/>
<FormFieldDescription Field="Well.LeaseName" Type="TextBox" Row="6"
Column="4" />
```

RemoveObservableCollections: Comma-delimited String

- Names of entity types that are on the many side of a one-to-many relationship with the Entity for which the UI Description is used that will not be displayed.

This allows you to remove (not display) a table on the many side of a one-to-many relationship. It is also possible to do this by simply not providing a UI.xml file for the table, but this method is preferred. For example, you might have a relationship of Construct->ConstructDates where Construct can have many ConstructDate records. If, for some reason, you do not want to display the dates, use this:

```
<RBDMS_UI_Description Name="WellConstruct" Label=" Construction
Records" RemoveObservableCollections="ConstructDates" xmlns="http://
schemas.microsoft.com/winfx/2006/xaml/presentation">
```

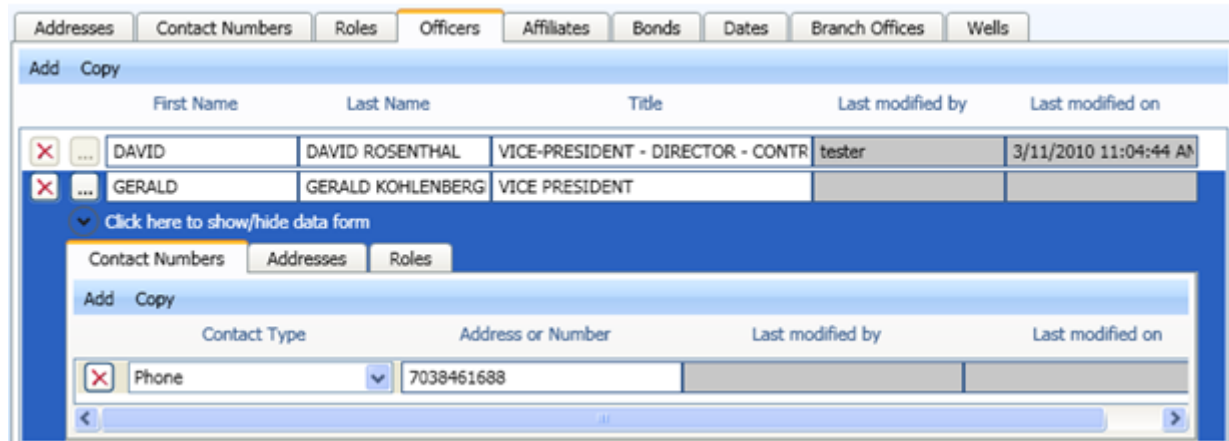
...and the ConstructDates tab disappears.

Form View

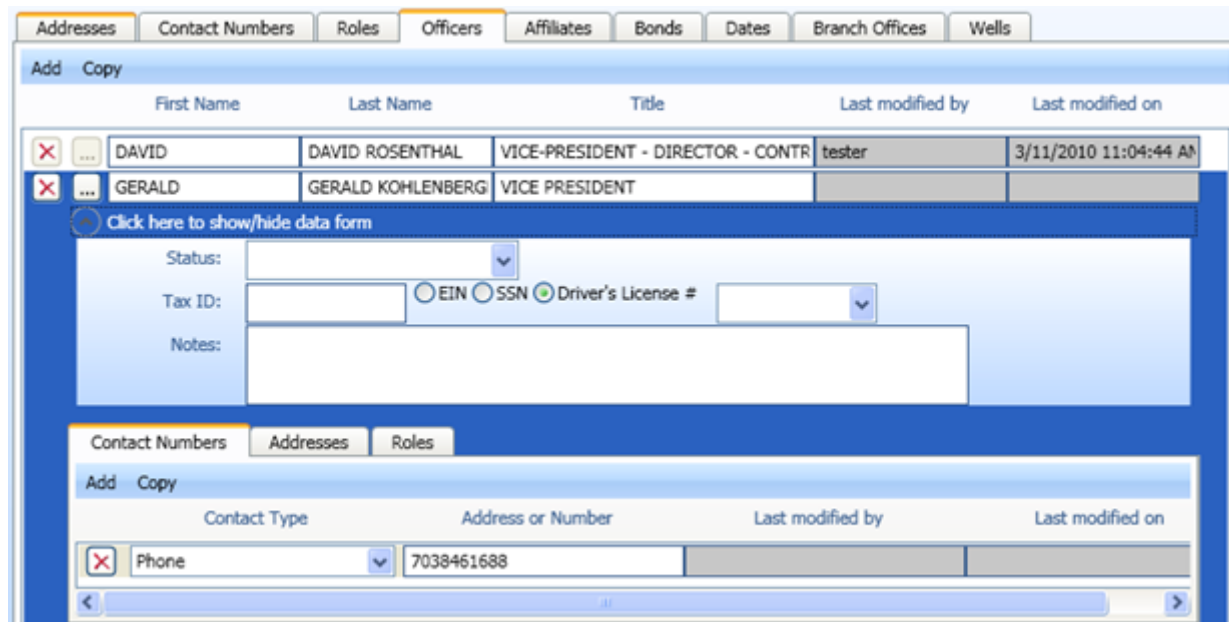
The Form View controls how the table is displayed as a form. The **RBDMS_UI_FormView** element controls overall aspects of the FormView. The **RBDMS_UI_FormView_Description** attributes are defined as follows:

- ☐ IsExpanded:
 - Boolean
 - True: Expand the subform initially.
 - False (default): Collapse the subform initially.

In a List view, a **Builder**  button expands a list entry so the user will see the form view of the list entry as soon as it is clicked. For example, EntityOfficers when IsExpanded = *False* (default) looks like this:



and if **IsExpanded** = *True*, looks like this when the **Builder**  button is pressed:



This can be useful if you want to make sure the user sees all of the data immediately, instead of having to press the  **Click here to show/hide data form** button.

- ☐ ShowExpander:
 - Boolean

- True (default): Show the expander button.
- False – do not show the expander button.

The ShowExpander is used to hide the  **Click here to show/hide data form** button. It is usually used to avoid showing this button in the topmost table.

FormGrid Definition

The Grid Definition defines the rows and columns of a form. It is exactly the same as a XAML grid definition. This allows you to define the size and number of rows and columns. Later, the row and column number are used in each element to place them on the form. Generally, for the topmost form, RBDMS.NET uses a fairly simple layout of five columns and as many rows as is needed:

```
<Grid Margin="5,0,5,10" >
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition Width="5"/>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
  </Grid.RowDefinitions>
</Grid>
```

Hint: Define more rows than you really need (maybe 3 times more), and skip rows when adding UI elements. This way, you can later easily add elements without having to renumber the row number for many UI elements. If there is nothing in a row, it collapses.

Form Field Descriptions

This is where you add the UI elements such as labels, text boxes, combo boxes, etc. The full specification of each of these UI Elements is in the topic [Dynamic UI](#)

Element Types. For example, a simple label and a text box that shows the contents of the *Operator* field in the Wells table:

```
<FormFieldDescription Label="Operator  
Number:" Type="Label" Row="2" Column="2" />
```

```
<FormFieldDescription Field="Operator" Type="TextBox" Row="2" Column="3"/>
```

Note the Row and Column that place the Label in Row 2, and Column 2, and the TextBox in Row 2, Column 3. The result is shown in this image:

Several attributes are common to all FormFieldDescription elements:

- ☐ Column:
 - Integer
 - The column the field goes in.
- ☐ ColumnSpan:
 - Integer
 - (Default=1)
 - The number of columns that the field spans.
- ☐ Row:
 - Integer
 - The Row the field goes in.
- ☐ RowSpan:
 - Integer
 - (Default=1)
 - The number of rows that the field spans.

An example of a fairly simple FormView section follows:

```
<RBDMS_UI_FormView_Description ShowExpander="False">  
  <Grid Margin="5,0,5,10" >  
    <Grid.ColumnDefinitions>  
      <ColumnDefinition/>  
      <ColumnDefinition/>
```

```

    <ColumnDefinition Width="5"/>
    <ColumnDefinition/>
    <ColumnDefinition/>
    <ColumnDefinition/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
    <RowDefinition/>
</Grid.RowDefinitions>
</Grid>
<FormFieldDescriptions>

```

```

    <FormFieldDescription Label="Nature of
Complaint" Type="Label" Row="1" Column="0" />

```

```

    <FormFieldDescription Field="Complaint" Type="MultiLineTextBox" Row="1" Column="1" ColumnSpan="4" ToolTip="Explain" >

```

```

        <FormFieldAdditionalAttributes>

```

```

            <FormFieldAdditionalAttribute TargetXName ="ValidatedTextBox" Height="50" />

```

```

            <FormFieldAdditionalAttribute TargetXName ="ValidatedTextBox" Width="500" /
>

```

```

        </FormFieldAdditionalAttributes>

```

```

    </FormFieldDescription>

```

```

    <FormFieldDescription Label="Water Body
Affected" Type="Label" Row="2" Column="0" />

```

```

    <FormFieldDescription Field="SpillWaterBody" Type="TextBox" Row="2" Column
="1" />

```

```

    <FormFieldDescription Label="Fish/Wildlife Kill
Reported" Type="Label" Row="4" Column="0"/>

```

```

    <FormFieldDescription Field="FishWildlifeKillReport" Type="ComboBox" Row="4"

```

```
Column="1" odpSource="odpYesNo"/>
```

```
<FormFieldDescription Label="Last Mod.  
User" Type="Label" Row="5" Column="0" />
```

```
<FormFieldDescription Field="ModifyUser"  
Type="ReadOnlyTextBox" Row="5" Column="1" />
```

```
<FormFieldDescription Label="Last Mod.  
Date" Type="Label" Row="5" Column="3" />
```

```
<FormFieldDescription Field="ModifyDate" Type="ReadOnlyTextBox" Row="5" C  
olumn="4" />
```

```
</FormFieldDescriptions>
```

```
</RBDMS_UI_FormView_Description>
```

This screen shot shows how it is displayed:

List View

The List View controls how a table (or LINQ class) is displayed as a list. The **RBDMS_UI_ListView** element controls overall aspects of the List View.

```
<RBDMS_UI_ListView_Description Header="Dates" AllowExpansion="False" Allo  
wCopy="False" AllowAdd="True" UpdateRightName="EntityDate"  
IsReadOnly="False" isSingleItem="False" >
```

Attributes of the **RBDMS_UI_ListView** element are as follows:

- ☐ Header:
 - String
 - The string that will be used in the tab for this ListView.
- ☐ AllowExpansion:
 - Boolean
 - True (Default): Each list row will have a "Click here to show/hide data form" button to show the form view for the row.
 - False: Does not have the  **Click here to show/hide data form** button.

- AllowCopy:
 - Boolean
 - True (Default): The **Copy** button appears in the list header. This allows you to copy the selected record to a new record.
 - False: The **Copy** button does not appear.
- AllowAdd:
 - Boolean
 - True (Default): The **Add** button appears in the list header. This allows you to add a new blank record.
 - False: The **Add** button does not appear.
- UpdateRightName:
 - String
 - An optional value that allows you to specify an RBDMS Right name that controls whether the entire tab will have update privileges. When specified, retrieves the security right and uses the *Update* property; otherwise, uses the *isReadOnly* value. For example, in the Mississippi RBDMS.NET application, a right named *ApprovePermit* allows only persons with this right the ability to change the status of permit records to *Approved*.
- IsReadOnly:
 - Boolean
 - True: The list is forced to be read-only. This overrides any other rights.
 - False (Default): Normal RBDMS C,R,U,D rights apply to the list unless otherwise specified at the control level.
- IsSingleItem:
 - Boolean
 - True: Instead of a list of items, just a single line (could even be a single column, e.g., a Datagrid control)
 - False (Default): Normal list behavior of showing a list all the items.

ListColumn Definition

These column XAML definitions define column number and width. These correspond to the XAML Grid.ColumnDefinitions. Note that no row definitions are needed, as the number of rows is determined by the number of records in the list. An example follows:

```
<ListColumnDefinitions>
  <ColumnDefinition MaxWidth="25" />
  <ColumnDefinition MaxWidth="25" />
  <ColumnDefinition MaxWidth="150" />
  <ColumnDefinition MaxWidth="150" />
```

```

<ColumnDefinition MaxWidth="150" />
<ColumnDefinition MaxWidth="200" />
<ColumnDefinition MaxWidth="150" />
<ColumnDefinition Width="*" />
</ListColumnDefinitions>

```

Hint: Buttons are usually in columns 25 units, but the rest of the columns can be variable sizes. The last column should be * to fill out the rest of the space in the list.

List Field Descriptions

Each column in the list should have an entry that specifies what will be displayed in that column. The full specification of each of these UI elements is in the [Dynamic UI Element Types](#) section below. However, several attributes are common to all **ListFieldDescription** elements:

- Label:
 - String
 - The label of the column.
- SortButton:
 - Boolean
 - True: Column is sortable.
 - False (default): Column is not sortable.
- InitialSortDirection:
 - String
 - Descending: The column is sorted initially in descending order.
 - Ascending: The column is sorted initially in ascending order.
- Column:
 - Integer
 - The column the field goes in.
- ColumnSpan:
 - Integer
 - (Default=1)
 - The number of columns that the field spans.

Example:

```

<ListFieldDescriptions>
  <ListFieldDescription Type="DeleteButton" Column="0" ToolTip="Delete this
event" />
  <ListFieldDescription Field="EntityDate" Label="Event
Date" Type="CalendarComboBox" Column="1" SortButton="True" InitialSortDirec
tion="Descending"/>

```

```

        <ListFieldDescription Field="TypeDate" Label="Event
Type" Type="ComboBox" Column="2" odpSource="odpEntityTypeDate" />

<ListFieldDescription Field="Comment" Label="Comment" Type="TextBox" Colu
mn="3" ColumnSpan="1" />
        <ListFieldDescription Field="ModifyUser" Label="Last modified
by" Type="ReadOnlyTextBox" Column="4" />
        <ListFieldDescription Field="ModifyDate" Label="Last modified
on" Type="ReadOnlyTextBox" Column="5" />
    </ListFieldDescriptions>

```

An example of a complete ListView section follows:

```

<RBDMS_UI_ListView_Description Header="Volumes">
    <ListColumnDefinitions>
        <ColumnDefinition MaxWidth="25" />
        <ColumnDefinition MaxWidth="25" />
        <ColumnDefinition MaxWidth="150" />
        <ColumnDefinition MaxWidth="150" />
        <ColumnDefinition MaxWidth="150" />
        <ColumnDefinition MaxWidth="150" />
        <ColumnDefinition MaxWidth="150" />
        <ColumnDefinition Width="*" />
    </ListColumnDefinitions>
    <ListFieldDescriptions>
        <ListFieldDescription Type="DeleteButton" Column="0" ToolTip="Delete this
record." />

<ListFieldDescription Type="FormExpanderButton" Column="1" ToolTip="Expand
this record." />
        <ListFieldDescription Field="SourceCode" Label="Source
Code" Type="ComboBox" Column="2" odpSource="odpIncidentVolumeSourceCod
e"/>
        <ListFieldDescription Field="OilReleased" Label="Oil
Released" Type="TextBox" Column="3" />
        <ListFieldDescription Field="OilRecovered" Label="Oil
Recovered" Type="TextBox" Column="4" />
        <ListFieldDescription Field="WaterReleased" Label="Water
Released" Type="TextBox" Column="5" />
        <ListFieldDescription Field="WaterRecovered" Label="Water
Recovered" Type="TextBox" Column="6" />
    </ListFieldDescriptions>
</RBDMS_UI_ListView_Description>

```

The following screen shot shows the display:

Add Copy		Source Code	Oil Released	Oil Recovered	Water Released	Water Recovered
		Code1	23	22	0	0
		Code2 Desc	0	0	0	0
		Code1	0	0	20	20

Dynamic UI Element Types

About Dynamic UI Element Types

The DynUI elements are the basic field elements used to display data fields, drop-down lists, buttons, etc. In general, the UI elements for both ListView and FormView are identical. However, sometimes slight differences are noted. Also, XAML styles are used to control the visual appearance of the control (see the the **Styles** subdirectory, file name **RbdmsControlStyles.xaml**).

In addition to the attributes associated with each Dynamic Element UI Type below you can specify **FormFieldAdditionalAttributes** for full access to the underlying XAML. The **TargetXName** must match the resultant type of the control. The Dynamic UI Element Type does not always match the resultant control type. For example, a TextBox Dynamic UI Type results in a ValidatedTextBox control. Where the resultant type differs, they are specified in the example below.

```
<FormFieldAdditionalAttributes>
```

```
<FormFieldAdditionalAttribute TargetXName="Button" Width="50"/>
```

```
<FormFieldAdditionalAttribute TargetXName="Button" HorizontalAlignmen  
t="Right"/>
```

```
</FormFieldAdditionalAttributes>
```

Attributes Common to All DynUI Elements

- ☐ Type:
 - String
 - The Type of the Element (e.g., Label, TextBox, etc.) These are shown in Bold in the element descriptions below.
- ☐ Name:
 - String
 - The name of the element. This can be used in code to reference this element. If not supplied, then the name is created automatically from the name of the field, plus a prefix in this manner:

```
Case "label"
```

```

        Return "lbl" & Regex.Replace(FD.@Label, "[^\w-]+", "")
    Case "textbox"
        Return "txt" & FD.@Field
    Case "checkbox"
        Return "chk" & FD.@Field
    Case "combobox"
        Return "cbo" & FD.@Field
    Case "readonlytextbox"
        Return "rotxt" & FD.@Field
    Case "pickandview"
        '//20110718 - DL - Mod to allow use of Field instead of
KeyField.
        Return "pav" & If(IsNothing(FD.@KeyField), FD.@Field,
FD.@KeyField)
    Case "calendarcombobox"
        Return "cal" & FD.@Field
    Case "multilinetextbox"
        Return "mltxt" & FD.@Field
    Case "reportgrid"
        Return "rpt" & FD.@FilterField

```

- ☐ Visibility:
 - String
 - Visible, Hidden or Collapsed
- ☐ Tag:
 - String
 - XAML Tag property
- ☐ Row: The row in the grid to place the control (unused in ListView).
- ☐ RowSpan: the number of rows to span in the grid with this control (unused in ListView).
- ☐ Column: The column in the grid to place the control.
- ☐ ColumnSpan: The number of columns to span in the grid with this control.
- ☐ Tooltip:
 - String
 - A string to display as the mouse pointer moves over the control. Sometimes, if this is not explicitly specified, then the contents of the control will automatically show as the tooltip.

Label

Label: Usually used to label input text boxes in form view. It is generally not used in ListView, but is mostly used only in FormView.

- Label:

- String
 - The text to be displayed.
- `Style="{StaticResource editorLabel}"`

Example:

```
<FormFieldDescription Label="Incident
Type:" Type="Label" Name="lblIncidentType" Row="4" Column="0"/>
```

Incident Type:

TextBox

TextBox: Resultant type ValidatedTextBox; used to enter and display many types of data fields.

- MaxLength
 - Integer
 - The maximum length that can be entered into the text box. Useful to limit typing too much into short fields.
- Field: The database field name that will be displayed/edited.
- Converter: A converter to be applied to the Field data when read and when written.
- ConverterParameter: A parameter to be supplied to the converter.
- Stringformat: A string format to be applied to the field data when read and written.
- TargetNullValue: A value to be used if the field value is Null.
- Style = editorTextBox (form) or ListeditorTextBox (List).
- Width: the Width of the text box.
- Tooltip: if not specified, the contents of the textbox are displayed.

Example:

```
<FormFieldDescription Field="SpillWaterBody" Type="TextBox" Row="2" Column
="1" />
```

Wasatchee Lake

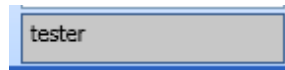
ReadOnly TextBox

ReadOnlyTextBox: Resultant type ValidatedTextBox. Always read-only.

- See [TextBox](#).
- Style = editorReadOnlyTextBox (form) or ListeditorReadOnlyTextBox (List).

Example:

```
<FormFieldDescription Field="ModifyUser"
Type="ReadOnlyTextBox" Row="5" Column="1" />
```



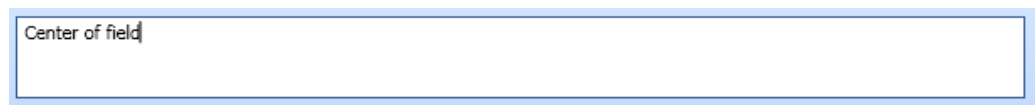
Multiline TextBox

MultiLineTextBox: Resultant type ValidatedTextBox. Has a style that allows multi-line entry for comment fields, etc.

- See [TextBox](#).
- Style = multilineEditorTextBox (form) or ListmultilineEditorTextBox (List).

Example:

```
<FormFieldDescription Field="Comment" Type="MultiLineTextBox" Row="14" Column="1" ColumnSpan="3" ToolTip="Notes." MaxLength="254" />
```



Hint: Often the size of a multi-line text box needs to be different than the default. This can be done in this manner:

```
<FormFieldDescription Field="Complaint" Type="MultiLineTextBox" Row="1" Column="1" ColumnSpan="4" ToolTip="Explain" >
```

```
<FormFieldAdditionalAttributes>
```

```
<FormFieldAdditionalAttribute TargetXName="ValidatedTextBox" Height="50" />
```

```
<FormFieldAdditionalAttribute TargetXName="ValidatedTextBox" Width="500" />
```

```
</FormFieldAdditionalAttributes>
```

```
</FormFieldDescription>
```

Giving this result (somewhat shorter):



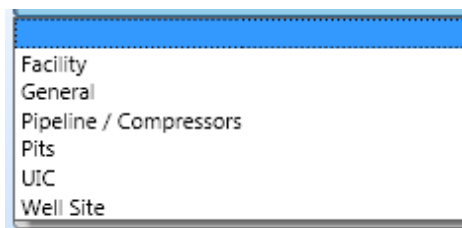
ComboBox

ComboBox: Drop-down combo box, normally used with Refcode fields to show a list of possible options.

- DisplayMemberPath: Defaults to *Description*.
- SelectedValuePath: Defaults to *Code*.
- odpSource: The name of the Data Provider that returns the Description, given the Refcode. These are in the Styles subdirectory, the **RbdmsDataProviders.xaml** file.
- Visibility:
 - String
 - Visible, Collapsed, Hidden
- Field: The database field.
- Converter: A converter to be applied to the Field data when read and when written.
- ConverterParameter: A parameter to be supplied to the converter.
- Tooltip: If not specified, the contents of the ComboBox are displayed.

Example:

```
<FormFieldDescription Field="IncidentType" Type="ComboBox" Name="cboIncidentType" Row="4" Column="1" odpSource="odpIncidentType"/>
```



The Refcodes lookup works as follows:

In the **RBDMSDataProviders.xaml** file in the **\Styles** folder, there is the following entry:

```
<ObjectDataProvider x:Key="odpIncidentType" ObjectType="{x:Type RbdmsDAL:RefCode}"
    MethodName="GetRefCodesByField">
    <ObjectDataProvider.MethodParameters>
        <system:String>IncidentType</system:String>
    </ObjectDataProvider.MethodParameters>
</ObjectDataProvider>
```

This in turn calls the *GetRefCodesByField* method in the **RbdmsDAL\Reference_BLL.vb** file. In this example, the string "IncidentType"

is passed to *GetRefCodesByField*. This method then looks up all the refcodes associated with "IncidentType" with this LINQ statement:

```
Dim q = From rc In rDC.RefCodes Select rc Where rc.Fld = fieldName
Order By rc.Description
```

PKey	Fld	Code	Description	Active	Tag	CodeValue	TranslateCode	ModifyDate
860	IncidentType	01	Well Site	True	NULL	0	NULL	2007-02-02 15:...
861	IncidentType	02	Facility	True	NULL	0	NULL	2007-02-02 15:...
862	IncidentType	03	UIC	True	NULL	0	NULL	2007-02-02 15:...
863	IncidentType	04	Pipeline / Compr...	True	NULL	0	NULL	2007-02-02 15:...
864	IncidentType	05	Pits	True	NULL	0	NULL	2007-02-02 15:...
865	IncidentType	06	General	True	NULL	0	NULL	2007-02-02 15:...

In the database, the Refcodes table has entries like these:

Note that the *Description* is what is displayed to the user (default *DisplayMemberPath*), and *Code* (default *SelectedValuePath*) is what is returned to the application, and also stored in the Incident table in the "IncidentType" field.

TimePicker

The Dynamic UI was recently expanded to include a TimePicker control from the [Xceed extended WPF toolkit](#), a third-party control set with royalty-free licensing for developers and users.

Properties:

- AllowSpin: Gets or sets a value indicating whether interacting with the spinner will result in increment/decrement operations.
- EndTime: Gets or sets the last time listed in the control's dropdown.
- Format: Gets or sets the time format displayed.
- FormatString: Gets or sets the format string used when Format is set to Custom.
- IsOpen: Gets or sets a value indicating whether the dropdown is open.
- ShowButtonSpinner: Gets or sets a value indicating whether the button spinner is shown.
- StartTime: Gets or sets the first time listed in the control's dropdown.
- TimeInterval: Gets or sets the time interval between the TimeItem items listed in the dropdown.
- Value: Gets or sets the DateTime value of the TimePicker.
- Watermark: Gets or sets the watermark to display in the control.
- WatermarkTemplate: Gets or sets the watermark template.

Example:

```
<FormFieldDescription Label="Time:" Type="Label" Name="lblInspectTime"
Row="8" Column="5" />
```

```

        <FormFieldDescription Field="InspectTime" Type="TimePicker"
Name="tpInspectTime" Width="95" ToolTip="Time of inspection"
VerticalAlignment="Center" TimeInterval="00:15:00.000"/>

```

```

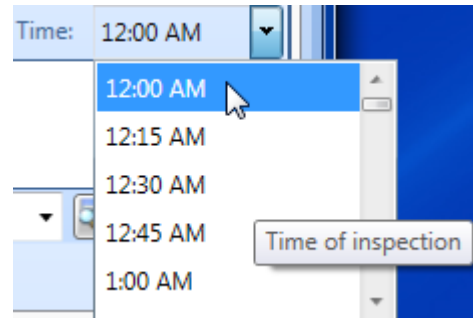
        <!--Converter="{StaticResource
dateTimeToTimeString}"-->

```

```

        <FormFieldDescription
Type="SimpleStackPanel" Name="spInspectDate"
Group="End"/>

```



Calendar ComboBox

CalendarComboBox: Resultant type toolkit: DatePicker. However if this is read-only (either by Right, or forced), then the resultant type is a Label with editorLabel or ListeditorLabel style.

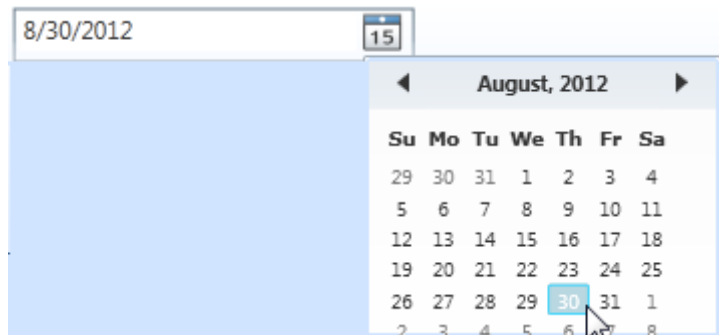
- Width
 - Integer
 - The desired width of the underlying DatePicker.
- Visibility:
 - String
 - Visible, Collapsed, Hidden
- HorizontalAlignment:
 - String
 - Center, Right, Left
- Field: The database field.
- Converter: converter to be applied to the Field data when read and when written.
- ConverterParameter: A parameter to be supplied to the converter.
- Tooltip: If not specified, the selected date is displayed.

Example:

```

<FormFieldDescription Field="InspectDate" Type="CalendarComboBox" Row="4"
Column="4" ToolTip="Date of inspection" VerticalAlignment="Center">
    </FormFieldDescription>

```



TextBlock

TextBlock: Corresponds to the TextBlock XAML element. (Not available in ListView.)

- Text – String – The text to be displayed as a TextBlock.
- `Style="{StaticResource editorTextBlock}"`

DeleteButton

DeleteButton: Resultant type ListDeleteButton. Deletes the currently selected record. Looks different in ListView and FormView.

- Visibility:
 - String
 - Visible, Collapsed, Hidden
- ImageName: The name of an image to use. If not specified, the default trashcan image is displayed.
- LabelText: The text on the button in Formview. If this is not specified, then the image is displayed.
- ToolTip: ToolTip, defaults to "Click to Delete Record" if not specified.
- RightName: Allows an RBDMS right to control whether the delete button is enabled. If the specified Right for the logged-on user has Read permissions, then the delete button is enabled.
- Margin: A margin can be specified, if not, the default is "2,2,2,2".

Example (ListView):

```
<ListFieldDescription Type="DeleteButton" Column="0" ToolTip="Delete this record." />
```



Example (FormView):

```

<FormFieldDescription Type="DeleteButton" Column="3" Row="23" LabelText="
Delete">
    <FormFieldAdditionalAttributes>

<FormFieldAdditionalAttribute TargetXName="ListDeleteButton" HorizontalAlignm
ent="Center" Width="100"/>
    </FormFieldAdditionalAttributes>
</FormFieldDescription>

```



FormExpanderButton

FormExpanderButton: Resultant type FormExpanderButton. This is the

Builder  button that expands a row in Listview. It is not used in FormView.

- Visibility:
 - String
 - Visible, Collapsed, Hidden
- LabelText: The text on the button in FormView.
- ToolTip: Tooltip, no default.
- `Style="{StaticResource editorFormExpanderButton}"`
- FormUIXMLFile: This allows a custom UI.xml file to be used instead of the default one.

Example:

```

<ListFieldDescription Type="FormExpanderButton" Column="1" ToolTip="Expand
this record." />

```



SaveButton

SaveButton: Resultant type SaveButton. Normally only used in FormView.

- Visibility:
 - String
 - Visible, Collapsed, Hidden
- LabelText – The text on the button in Formview. If not specified, then a “diskette” image is displayed instead.
- ToolTip – Tooltip, no default.

Example:

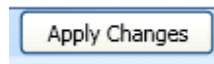
```
<FormFieldDescription Type="SaveButton" Column="1" Row="23" LabelText="Apply Changes">
```

```
<FormFieldAdditionalAttributes>
```

```
<FormFieldAdditionalAttribute TargetXName ="SaveButton" HorizontalAlignment="Center" Width="100"/>
```

```
</FormFieldAdditionalAttributes>
```

```
</FormFieldDescription>
```



Print Button

PrintButton: Used for printing operations in Form View or List View.

- LabelText: The text on the button. If not specified, then a "Printer" image is displayed instead.
- ToolTip: ToolTip, defaults to *Click to Print*.
- Name **btnPrint**: Cannot be changed. Use this name to reference this control in code. Clicks will generate RbdmsWpfControlsBase.PrintButton.PrintEvent events in code.

Example:

```
<ListFieldDescription Type="PrintButton" Column="2" ToolTip="Print Letters." />
```



Button

Button: A generic button that does not have any default behavior and has to be linked to code. Note that a button will not be visible if the form is not editable.

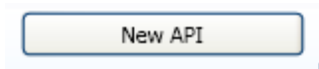
Not typically used in ListView.

- Name: The XAML name that code can refer to.
- Content: The text that appears in the button.
- ToolTip: No default.
- Margin: Defaults to 2,2,2,2.
- RightName:
 - If present, the DynUI will retrieve this right for the current user and override all other security settings on the DynUI Element with appropriate settings based on the C,R,U,D of the right.
 - If the Right has Read permissions, then it is combined with the other permissions.

- If the Right does not have Read permission, then the button is disabled, no matter what other rights are present.
- Height: The height of the control.
- Width: The width of the control.
- ImageSource: Allows an image to be used instead of the Content.
- Content: If ImageSource is not specified, then this is displayed as the button Content.

Example:

```
<FormFieldDescription Name="btnCreateAPI" Type="Button" Content="New API" Row="0" Column="2" ToolTip="Create and reserve an API." Margin="13,2,2,2" />
```



NavigateButton

NavigateButton: Used to open a new main tab to show a main table like Well, Entity, Bond etc. The EditorFullName specifies the type of editor to open, and the LinkIDSource is the field that supplies the PKey of the record that will be displayed. The TitleSource specifies the field that will provide the text that will be shown in the tab of the new window.

- ToolTip: No default.
- LinkIDSource: Field that supplies the Pkey of the record that will be displayed in the specified editor.
- EditorTitleSource:
 - String
 - Field that will provide the text that will be shown in the tab of the new window.
- EditorFullName: Specifies the type of editor to open.

Example:

```
<ListFieldDescription Type="NavigateButton" Column="0" ToolTip="View this Inspection"
```

```
LinkIDSource="PKey" EditorFullName="RbdmsWpfControlsBase.InspectEditor"
EditorTitleSource="InspectDate"/>
```



CheckBox

CheckBox: Used for Boolean fields.

- **IsThreeState:**
 - Boolean.
 - If set to *True*, allows the check box to have the three possibilities of *Checked*, *Unchecked*, and *Neither*.
 - Useful if you need to set a field to *Null*. Note that if the field is *Null*, the check box initially will be in the *Neither* state regardless of this setting. However, the user can set to *Checked* or *Unchecked* only if this is set to *False*.
- **ToolTip:**
 - String.
 - Specifies a tooltip to show when the mouse is over the check box.
- **Field:** The field to display.
- **Converter:** A converter to be applied to the Field data when read and when written.
- **ConverterParameter:** Parameter to be supplied to the converter.

Example:

```
<FormFieldDescription Field="DoNotAllowWellTransfer" Type="CheckBox" Row="9" Column="1" ToolTip="Well Transfer Blocked if this is checked"/>
```



Pick-and-View

PickAndView: A composite control used widely in RBDMS.NET to allow the user to select and view items from another table. For example, in the Inspection editor, a PickAndView is used to select an Inspector from a list of inspectors. While the name of the Inspector is shown, only the PKey of the inspector is stored in the PickAndView field.

- **Width:** The width of the control.
- **ShowViewButton:**
 - Boolean.
 - True (default): Show the View button.
 - False: Do not show the View button.
- **ShowPickButton:**
 - Boolean.
 - True (default): Show the Pick button.
 - False: Do not show the Pick button.
- **ShowClearButton:**
 - Boolean.
 - True (default): Show the Clear button.
 - False: Do not show the Clear button.

- AllowTypeAhead:
 - Boolean.
 - True (default): Enable the type ahead functionality. When the user types at least 3 characters, matching entries will appear in a dropdown.
 - False: Do not enable the type ahead. This may be useful if there are too many entries and the search response is too slow.
- EntityKey:
 - String.
 - Name of the selected primary key data.
- KeyField:
 - String
 - Name of LINQ property that contains the foreign key to the desired record in the foreign table.
- MaxSelectedRecords:
 - Integer
 - If greater than 0 is used to limit the number of selected records to MaxSelectedRecords.
- TargetRecordType:
 - String
 - Type of Pick and View desired
 - Entity: The most commonly used. Used in conjunction with RoleCode.
 - Docket
 - Permit
 - Receipt: Actually a receipt instrument.
 - Field
 - Formation
 - Pool
 - Well: A single well
 - Construct: A single lateral from a well
 - Subscription
- RoleCode:
 - String
 - Used to filter the Entity by its Role. Where you are looking up an Entity, the RoleCode matches any Entity that has that Role.

Example:

```
<FormFieldDescription KeyField="InspectorKey" Type="PickAndView" Row
="12" Column="1" VerticalAlignment="Center" TargetRecordType="Inspector" RoleCode="FLDINSP" />
```



In this case, the RoleCode is FLDINSP, so the entities to be displayed either by type-ahead or by the **Builder**  button will be only be those that have that role.

SimpleStackPanel

SimpleStackPanel: Resultant type StackPanel. Allows you to group several elements into a group. This group then behaves as a single element in respect to positioning on a form. To form a group, you need two of SimpleStackPanels, one at the beginning of the group with Group="Start," and one at the end with Group="End." Typically only used in FormView.

- Visibility:
 - String
 - Visible, Collapsed, Hidden
- Orientation: Vertical or Horizontal stacking.
- Group: Set to "Start" for the beginning SimpleStackPanel, and "End" for the final SimpleStackPanel (see the example).

Example:

```
<FormFieldDescription Type="SimpleStackPanel" Name="stpTaxID" Group="Start" Row="7" Column="1" ColumnSpan="6" Orientation="Horizontal" />
```

```
<FormFieldDescription Field="DecryptedTaxID" Name="txtDecryptedTaxID" Type="TextBox" >
```

```
<FormFieldAdditionalAttributes>
```

```
<FormFieldAdditionalAttribute TargetXName="ValidatedTextBox" Width="100"/>
```

```
</FormFieldAdditionalAttributes>
```

```
</FormFieldDescription>
```

```
<FormFieldDescription
```

```
Type="RadioButton" Name="rbTaxID_EIN" IsChecked="{Binding Path=IsTypeEIN,Mode=TwoWay}" Content="EIN" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" />
```

```
<FormFieldDescription
```

```
Type="RadioButton" Name="rbTaxID_SSN" IsChecked="{Binding Path=IsTypeSSN,Mode=TwoWay}" Content="SSN" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" />
```

```
<FormFieldDescription
```

```
Type="RadioButton" Name="rbTaxID_DL" IsChecked="{Binding
```

```

Path=IsTypeDL,Mode=TwoWay}" Content="Driver's License #
" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" /
>

<FormFieldDescription Name="cbDLNState" Field="DLNState" Type="Co
mboBox" odpSource="odpState" >
    <FormFieldAdditionalAttributes>

<FormFieldAdditionalAttribute TargetXName="ComboBox" Width="100"/>
    </FormFieldAdditionalAttributes>
</FormFieldDescription>

<FormFieldDescription Type="SimpleStackPanel" Name="spTaxID" Group
="End"/>

```

ReportGrid

ReportGrid: Simple way to include a report in a subtab. Normally used only in ListView with isSingleItem="True."

- Visibility:
 - String
 - Visible, Collapsed, Hidden
- Report: Name of the report.
- FilterField: Source of the value used to filter the report.
- FilterString: A string that is used to filter the report in conjunction with FilterField.

Example:

```

<RBDMS_UI_Description Name="WellInjection" Label="Injection" xmlns=
"http://schemas.microsoft.com/winfx/2006/xaml/presentation">
    <!-- IsSingleItem makes a single line instead of a list of items. Here we
    want a single datagrid. -->

<RBDMS_UI_ListView_Description Header="Injection" isReadOnly="True"
isSingleItem="True">
    <ListColumnDefinitions>
        <ColumnDefinition />
    </ListColumnDefinitions>
    <ListFieldDescriptions>

<ListFieldDescription Type="ReportGrid" Name="rptInjection" Column="0
" Report="WellInjectionDynUI.xml" FilterString="WellKey=" FilterField="P
Key" />
    </ListFieldDescriptions>
</RBDMS_UI_ListView_Description>

```

```
</RBDMS_UI_Description>
```

Generic XAML Elements

If the Type of the ListFieldDescription or the FormFieldDescription is not one of the above RBDMS defined types, the DynUI attempts to interpret it as a generic XAML control. All attributes are simply passed through untouched. For example, an XAML Expander control:

```
<FormFieldDescription Type="Expander" Header="Click here to  
expand" IsExpanded="False" Grid.Row="5" Grid.Column="0" xmlns  
s="http://schemas.microsoft.com/winfx/2006/xaml/presentation" />
```

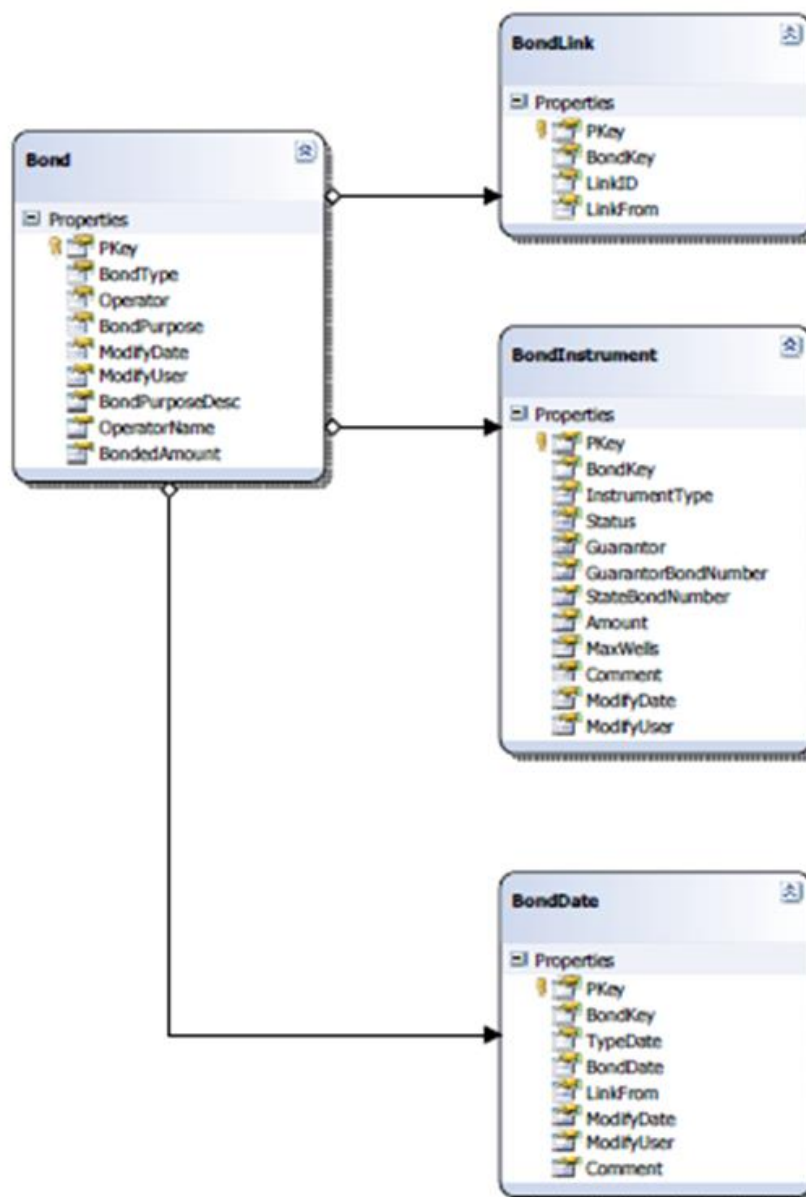
Note: The xmlns needs to be specified as shown in the example.

VisualBasic.NET Programmer's Reference

When Recompiling Is Needed

While all of the above .xml files can be manipulated without re-compiling the RBDMS.NET code, a number of actions will require recompiling, such as adding additional tables or fields or relationships to the O/R designer (the .dbml files). As mentioned above, the schema (actually the .dbml files) define the LINQ objects, and subsequently, the xxxx_UI.xml files.

The O/R designer allows you to easily add or remove tables and relationships visually. The resulting visual representation can be useful to see how all the tables are related. However, this visual relationship also generates the LINQ To SQL objects that are used within RBDMS.NET. Here is a relatively simple O/R design:



For example, suppose that we would like to add an additional table called BondFine that shows the fines that have been paid for this bond (purely fictitious). In the database there is a table BondFine that has a foreign key named

BondKey that contains the Bond that a BondFine is attached to by a one-to-many relationship. Here is an SQL script that will create this table:

```
USE [RBDMS_PROD]
```

```
GO
```

```
/****** Object: Table [dbo].[BondFine]   Script Date: 04/11/2012  
11:43:35 *****/
```

```
IF
```

```
EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[d  
bo].[BondFine]') AND type in (N'U'))
```

```
IF
```

```
EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id = OBJECT_ID  
(N'[dbo].[FK_BondFine_Bond]') AND parent_object_id  
= OBJECT_ID(N'[dbo].[BondFine]'))
```

```
ALTER TABLE [dbo].[BondFine] DROP CONSTRAINT [FK_BondFine_Bond]
```

```
GO
```

```
IF
```

```
EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[d  
bo].[BondFine]') AND type in (N'U'))
```

```
DROP TABLE [dbo].[BondFine]
```

```
GO
```

```
/****** Object: Table [dbo].[BondFine]   Script Date: 04/11/2012  
11:43:35 *****/
```

```
SET ANSI_NULLS ON
```

```
GO
```

```
SET QUOTED_IDENTIFIER ON
```

```
GO
```

```
CREATE TABLE [dbo].[BondFine](
```

```
    [BondKey] [int] NULL,
```

```
    [FinePaid] [money] NULL,
```

```
    [ModifyDate] [date] NULL,
```

```
    [ModifyUser] [nvarchar](50) NULL,
```

```
    [Pkey] [int] IDENTITY(1,1) NOT NULL
```

```
    CONSTRAINT [PK_BondFine] PRIMARY KEY CLUSTERED
```

```
(
```

```
    [Pkey] ASC
```

```

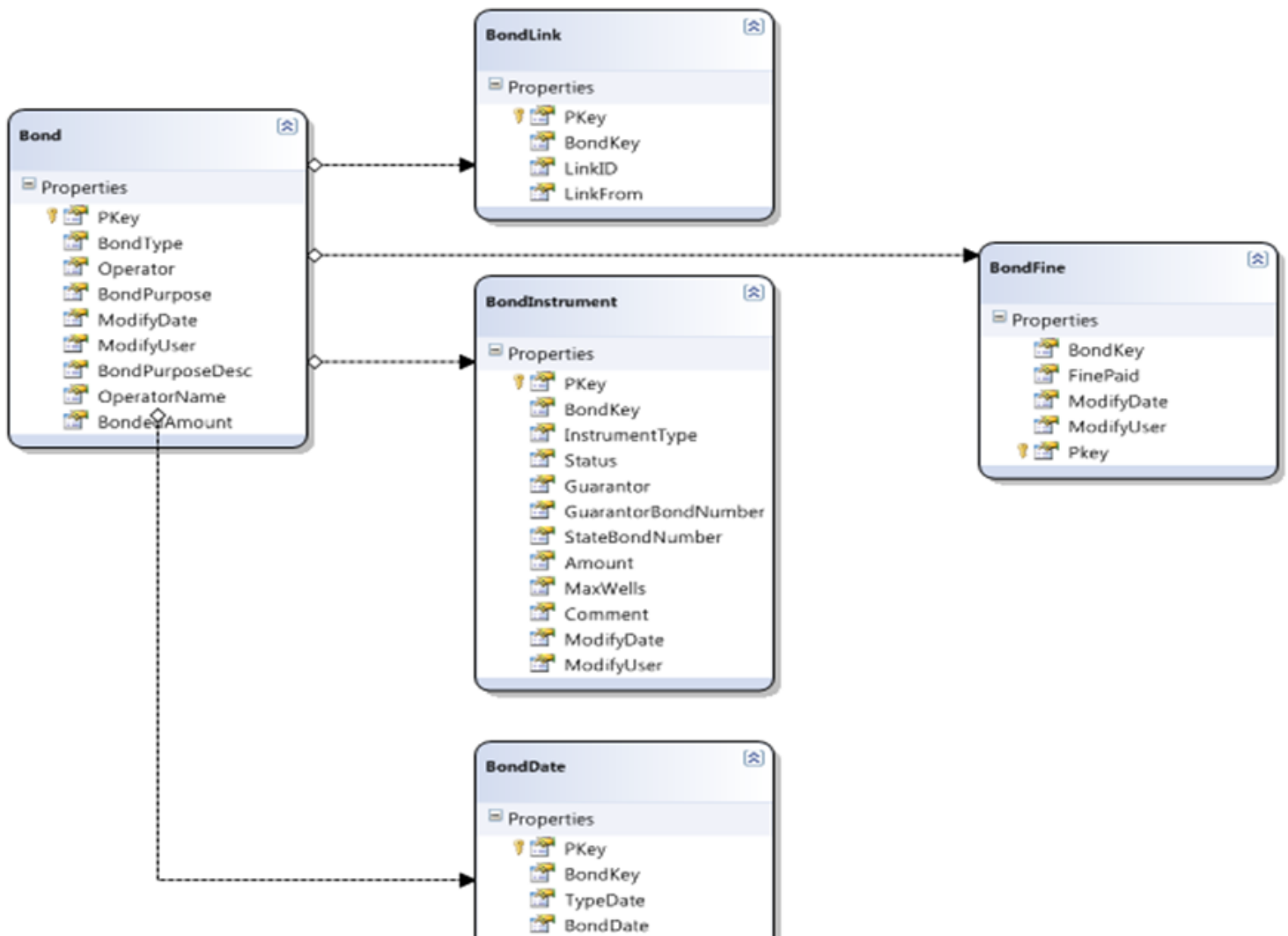
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE
= OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS
= ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

ALTER TABLE [dbo].[BondFine]
WITH CHECK ADD CONSTRAINT [FK_BondFine_Bond]
FOREIGN KEY([BondKey])
REFERENCES [dbo].[Bond]([PKey])
GO

```

Once the table was created, you would perform the following steps:

1. Open the **Bond_DataClasses.dbml** in the RbdmsDAL project.
2. Drag and drop the BondFine table from the Server Explorer – Data Connections to the O/R designer. Since there is already a Foreign key constraint, the O/R designer will automatically set up a one-to-many relationship to the Bond table. The resulting O/R diagram looks like this:



The next step is to add a small entry to the **Bond.BLL.vb** file to make sure the LINQ objects inherit from BaseBusiness2 (has a number of helper methods that make RBDMS.NET work properly). In this case:

```

Public Class BondFine
    Inherits BaseBusiness2

    Private Sub BondDate_PropertyChanged(ByVal sender
    As Object, ByVal e
    As System.ComponentModel.PropertyChangedEventArgs)
        Handles Me.PropertyChanged
            Me.IsDirty = True
            ExecuteValidationChecks()
        End Sub
    End Class
  
```

```

End Sub
Private Sub ExecuteValidationChecks()
End Sub

```

```
End Class
```

Next, create the UI XML file. It will be named **BondFines_UI.xml** and located with all the other UI.xml files in the **DynamicUIXML** subdirectory. Here is a simple XML file that will display the elements in the BondFine table for a particular Bond.

```

<RBDMS_UI_Description Name="BondFines" Label="Fines" xmlns="http://
/schemas.microsoft.com/winfx/2006/xaml/presentation">
  <RBDMS_UI_FormView_Description ShowExpander="False">
    <Grid Margin="5,0,5,10" >
      <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition Width="5"/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
      </Grid.RowDefinitions>
    </Grid>
    <FormFieldDescriptions>

      <FormFieldDescription Label="Last Mod.
User" Type="Label" Row="5" Column="0" />
      <FormFieldDescription Field="ModifyUser"
Type="ReadOnlyTextBox" Row="5" Column="1" />
    </FormFieldDescriptions>
  </RBDMS_UI_FormView_Description>
</RBDMS_UI_Description>

```

```

</FormFieldDescriptions>
</RBDMS_UI_FormView_Description>
<RBDMS_UI_ListView_Description Header="Fines">
  <ListColumnDefinitions>
    <ColumnDefinition MaxWidth="25" />
    <ColumnDefinition MaxWidth="25" />
    <ColumnDefinition MaxWidth="150" />
    <ColumnDefinition MaxWidth="150" />
    <ColumnDefinition Width="*" />
  </ListColumnDefinitions>
  <ListFieldDescriptions>

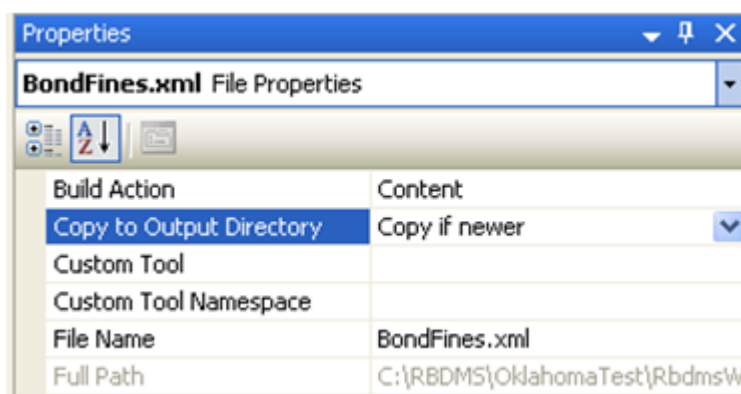
    <ListFieldDescription Type="DeleteButton" Column="0" ToolTip="Delete
this record." />

    <ListFieldDescription Type="FormExpanderButton" Column="1" ToolTip="
Expand this record." />

    <ListFieldDescription Field="FinePaid" Label="Amount" Type="TextBox" C
olumn="2" />
    <ListFieldDescription Field="ModifyDate" Label="Last modified
on" Type="ReadOnlyTextBox" Column="3" />
  </ListFieldDescriptions>
</RBDMS_UI_ListView_Description>
</RBDMS_UI_Description>

```

Hint: The properties of the **xxxx_UI.xml** file have to be set to Content, Copy if newer in order to end up in the right place if you are using content files, or leave as Resource if you wish to save the data into Resources.



Running the program gives us the expected result, a new tab **Fines** with the fields laid out as specified:

Adding, Editing, or Deleting Business Logic

The [BondFine](#) class **Bond.BLL.vb** is also where you can add any validation logic. Visual Studio.NET and **BaseBusiness2** take care of most of the basic logic needed to make everything work properly.

For example, we can allow fines that are only less than \$100, so anything entered above amount this will cause an error. To do this, modify the BondFine *ExecuteValidationChecks* like this:

```
Private Sub ExecuteValidationChecks()
    If FinePaid > 100.0 Then
        Me.AddError("FinePaid", "All fines must be less than or equal to
$100.")
    Else
        Me.RemoveError("FinePaid")
    End If
End Sub
```

The result is shown below:

Mousing over the exclamation mark produces the words "All fines must be less than or equal to \$100." The code examines the value of *FinePaid*, and calls *AddError* or *RemoveError* depending on the result.

- *AddError*: The first parameter is the field name, the second is the error message.
- *RemoveError*: The only parameter is the field name.

There are also some built-in validation checks with built-in messages you can use. These follow this pattern and are defined in

\RbdmsDAL\DataValidationRules.vb:

- *IsEmailValid*
- *IsRequiredFieldPresent*
- *IsRequiredStringPresent*
- *IsInteger*
- *IsPosInteger*
- *IsPosNumeric*

An example of their use is shown below:

```
Private Sub ExecuteValidationChecks()
    If Not DataValidationRules.IsRequiredFieldPresent(Me.Guarantor)
Then
        Me.AddError("Guarantor",
DataValidationRules.RequiredFieldPresentText)
    Else
        Me.RemoveError("Guarantor")
    End If
End Sub
```

Change Tracking

One more common programming tasks is Change Tracking. Whenever the user saves a record, it is possible to track any changes made in the record. You can select some fields to track but not to track others. Tracked changes typically create a new Date record in a related table. For example, in our example of [BondFines](#), you may wish to track the *FinePaid* field to record when this has been changed. The following example steps through this procedure.

First of all, there needs to be a table (called *BondFineDate* in this example) with a

one-to-many relationship with the BondFine table (i.e., for a single BondFine, there can be many BondFineDate records). Create this table with the following SQL script:

```
IF
EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id = OBJECT_ID
(N'[dbo].[FK_BondFineDate_BondFine]') AND parent_object_id
= OBJECT_ID(N'[dbo].[BondFine]'))
ALTER TABLE [dbo].[BondFineDate]
DROP CONSTRAINT [FK_BondFineDate_BondFine]
GO
```

```
/****** Object: Table [dbo].[BondFineDate]   Script Date: 04/12/2012
22:47:00 *****/
```

```
IF
EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[d
bo].[BondFineDate]') AND type in (N'U'))
DROP TABLE [dbo].[BondFineDate]
GO
```

```
/****** Object: Table [dbo].[BondFineDate]   Script Date: 04/12/2012
22:47:00 *****/
```

```
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
```

```
SET ANSI_PADDING ON
GO
```

```
CREATE TABLE [dbo].[BondFineDate](
    [PKey] [int] IDENTITY(1,1) NOT NULL,
    [BondFineKey] [int] NULL,
    [Event] [varchar](20) NULL,
    [EventDate] [datetime] NULL,
    [Description] [varchar](max) NULL,
    [ModifyDate] [datetime] NULL,
    [ModifyUser] [varchar](30) NULL,
    CONSTRAINT [PK_BondFineDate] PRIMARY KEY CLUSTERED
(
    [PKey] ASC
```

```

)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE
= OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS
= ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO

ALTER TABLE [dbo].[BondFineDate] WITH CHECK ADD
CONSTRAINT [FK_BondFineDate_BondFine] FOREIGN KEY([BondFineKey])
REFERENCES [dbo].[BondFine] ([PKey])
ON UPDATE CASCADE
ON DELETE CASCADE
GO

ALTER TABLE [dbo].[BondFineDate]
CHECK CONSTRAINT [FK_BondFineDate_BondFine]
GO

```

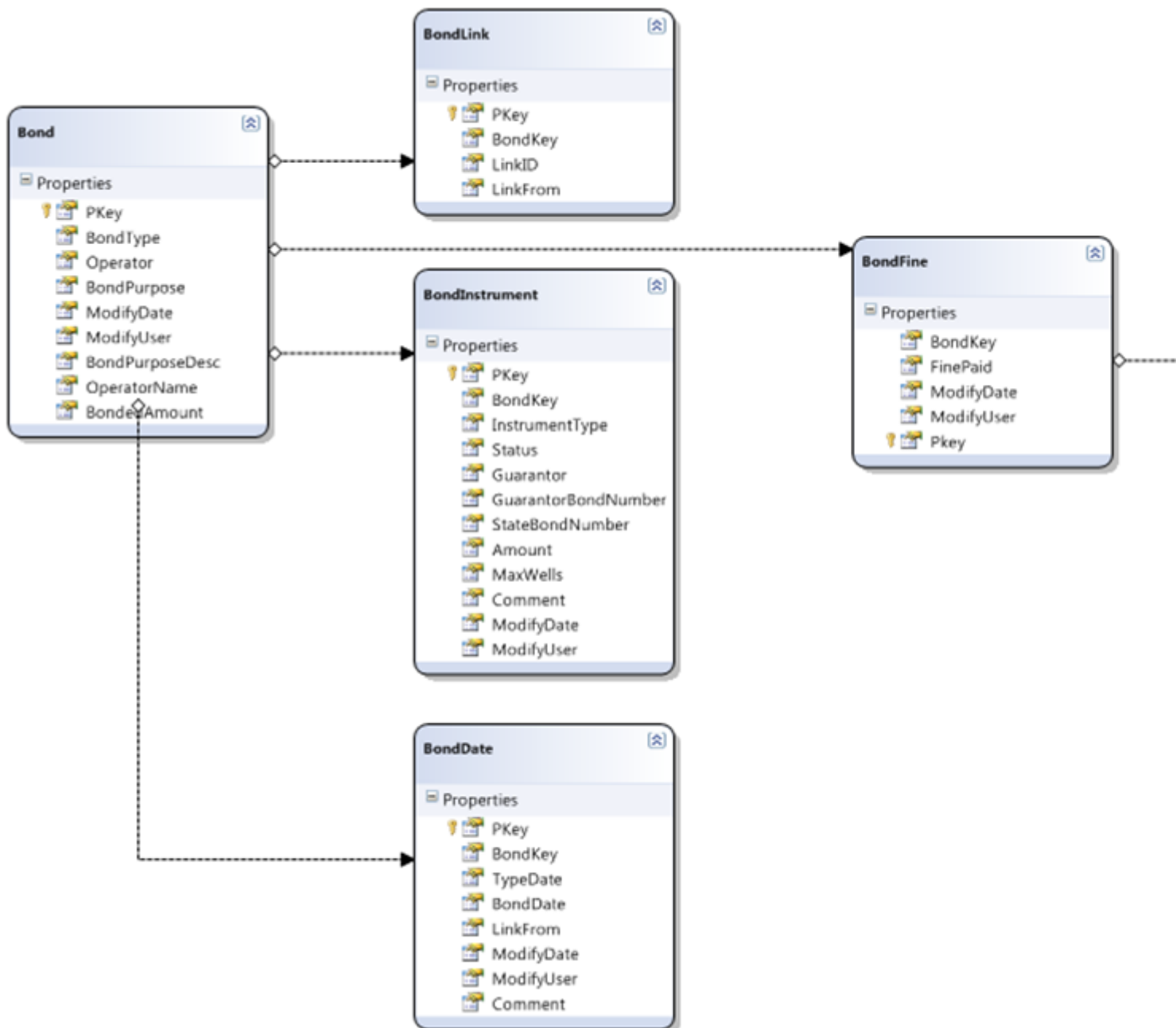
In this case, the table contains these fields:

```

[PKey] [int] IDENTITY(1,1) NOT NULL,
[BondFineKey] [int] NULL,
[Event] [varchar](20) NULL,
[EventDate] [datetime] NULL,
[Description] [varchar](max) NULL,
[ModifyDate] [datetime] NULL,
[ModifyUser] [varchar](30) NULL,

```

As before, we need to add this new table into the O/R designer. Open the **Bond_DataClasses.dbml** in the RbdmsDAL project. Next, drag and drop the BondFineDate table from the Server Explorer – Data Connections to the O/R designer. The resulting O/R diagram now looks like this:



Next, add the new class to the **Bond.BLL.vb** as before:

```
Public Class BondFineDate
    Inherits BaseBusiness2
```

```
Private Sub BondFineDate_PropertyChanged(ByVal sender As Object,
    ByVal e As System.ComponentModel.PropertyChangedEventArgs)
```

```

Handles Me.PropertyChanged
    Me.IsDirty = True
    ExecuteValidationChecks()
End Sub
Private Sub ExecuteValidationChecks()
End Sub

End Class

```

Finally, create a **BondFineDates_UI.xml** so we can see the contents of the table in the program:

```

<RBDMS_UI_Description Name="BondFineDates" Label="Fine
Dates" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentati
on">
    <!--AllowExpansion="false" prevents the form view from showing.-->

<RBDMS_UI_ListView_Description Header="Dates" AllowExpansion="False
">
    <ListColumnDefinitions>
        <ColumnDefinition Width="25"/>
        <ColumnDefinition MinWidth="140" Width="140" />
        <ColumnDefinition MinWidth="140" Width="140" />
        <ColumnDefinition MinWidth="140" Width="140"/>
        <ColumnDefinition MinWidth="140" Width="140"/>
        <ColumnDefinition MinWidth="140" Width="*/>
    </ListColumnDefinitions>
    <ListFieldDescriptions>

<ListFieldDescription Type="DeleteButton" Column="0" ToolTip="Delete
this event" />
        <ListFieldDescription Field="EventDate" Label="Event
Date" Type="CalendarComboBox" Column="1" SortButton="True" InitialS
ortDirection="Descending"/>
        <ListFieldDescription Field="Event" Label="Previous
Value" Type="TextBox" Column="2" />
        <ListFieldDescription Field="Description" Label="Event
Description" Type="TextBox" Column="3" ColumnSpan="1" />
        <ListFieldDescription Field="ModifyUser" Label="Last modified
by" Type="ReadOnlyTextBox" Column="4" />
        <ListFieldDescription Field="ModifyDate" Label="Last modified
on" Type="ReadOnlyTextBox" Column="5" />
    </ListFieldDescriptions>

```

```

</RBDMS_UI_ListView_Description>
</RBDMS_UI_Description>

```

Next, for the change tracking. We wish to put the previous value of the FinePaid into the *Event* field, and the name of the field being tracked into the *Description* field.

Add the OnLoaded method to BondFine as shown below:

```

Public Class BondFine
    Inherits BaseBusiness2

    Private Sub BondDate_PropertyChanged(ByVal sender As Object,
ByVal e As System.ComponentModel.PropertyChangedEventArgs)
        Handles Me.PropertyChanged
            Me.IsDirty = True
            ExecuteValidationChecks()
    End Sub

    Private Sub ExecuteValidationChecks()
        If FinePaid > 100.0 Then
            Me.AddError("FinePaid", "All fines must be less than or equal to
$100")
        Else
            Me.RemoveError("FinePaid")
        End If
    End Sub

    Private Sub OnLoaded()
        ChangeList = New ChangeListClass(Me)
        ChangeList.TargetTable = "BondFineDates"
        ChangeList.DateColumn = "EventDate"
        ChangeList.FieldNameColumn = "Description"
        ChangeList.PreviousDataColumn = "Event"
        ChangeList.FieldsToMonitor = New Dictionary(Of String, String)
        ChangeList.FieldsToMonitor.Add("FinePaid", FinePaid.Value)
    End Sub
End Class

```

The ChangeList class has the following properties that need to be set:

- d.ChangeList.TargetTable = "BondFineDates": TargetTable is the table to put the change tracking information. Note that this is pluralized.
- d.ChangeList.DateColumn = "EventDate": The date/time stamp of when this change occurred.
- d.ChangeList.FieldNameColumn = "Description": The field in the Target Table to put the name of the field that is changing.

- `d.ChangeList.PreviousDataColumn = "Event"`: The field in the Target Table to put the previous value of the field that is changing.
- `d.ChangeList.FieldsToMonitor = New Dictionary(Of String, String)`: `FieldsToMonitor` is a Dictionary type.
- `d.ChangeList.FieldsToMonitor.Add("FinePaid", FinePaid.Value)`: Add a dictionary entry describing the field to be tracked. The first entry is the name of the field. The second entry is for a starting value to compare against the field. Use the current value of `FinePaid`.

Finally, there is the *ChangeTrackerTrackPreviousValue* application setting. If this is set to true, then the previous value is saved in the `PreviousDataColumn`. Otherwise, the current value is saved. Here is a sample result after making two changes and saving the record:

The screenshot displays a web application interface for managing 'Fines'. At the top, there are tabs for 'Dates' and 'Fines'. Below the tabs, there are buttons for 'Add' and 'Copy'. The main form shows the 'Amount' as 65.00 and the 'Last modified on' date as 4/13/2012 12:43:07 PM. Below this, there is a field for 'Last Mod. User' with the value 'tester'.

Below the form, there is a table showing the history of changes. The table has columns for 'Event Date', 'Previous Value', 'Event Description', 'Last modified by', and 'Last modified on'.

Event Date	Previous Value	Event Description	Last modified by	Last modified on
4/13/2012	44.1100	FINEPAID	tester	4/13/2012 12:42:55 PM
4/13/2012	55.00	FINEPAID	tester	4/13/2012 12:43:07 PM

Note: The change tracking does not track changes made during the initial creation and editing of the `FinePaid` record.